



零基础搭建 量化投资系统

以Python为工具

何战军 杨茂龙 何天琦 编著

电子工业出版社

Publishing House of Electronics Industry

北京•BEIJING



电子工业出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

内 容 简 介

本书从初学者的角度出发,通过通俗易懂的语言,详细介绍了量化投资分析、机器学习、NLP 自然语言处理(聊天机器人设计)、网络爬虫等应用知识,书中所有知识点都结合具体实例进行讲解,可以使读者轻松领会 Python 程序开发的精髓,让零基础的读者轻松跨入编程领域。本书还通过一个完整的项目案例,帮助读者独立搭建量化分析交易平台。

本书适合 Python 语言零基础的学生、用 Python 语言编写量化交易策略的开发人员,以及从事数据建模框架和量化分析交易框架的设计人员与机器学习、NLP 自然语言处理、网络爬虫应用开发的人员阅读。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

零基础搭建量化投资系统:以 Python 为工具 / 何战军, 杨茂龙, 何天琦编著. —北京:电子工业出版社, 2019.12

(量化交易丛书)

ISBN 978-7-121-37620-7

I. ①零… II. ①何… ②杨… ③何… III. ①软件工具-程序设计 IV. ①TP311.561

中国版本图书馆 CIP 数据核字(2019)第 219953 号

责任编辑:黄爱萍

印 刷:

装 订:

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱

邮编:100036

开 本:787×980 1/16 印张:27.5 字数:530 千字

版 次:2019 年 12 月第 1 版

印 次:2019 年 12 月第 1 次印刷

定 价:99.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888, 88258888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式:010-51260888-819, faq@phei.com.cn。



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

前 言

本书主要介绍有关 Python 3.7 的使用，Python 3 是发展趋势，由于 Python 2 对中文支持不友好等局限性会被逐步淘汰。

本书是金融投资和量化分析软件开发的入门书籍，其重点介绍金融数据的处理和投资分析技术的程序实现、K 线图形和技术指标图形的程序绘制与框架开发，以及 Tkinter 窗口布局设计等，目的是使读者能够独立搭建自己的金融投资分析量化平台。

通过对本书的学习，读者能在 Windows 平台上搭建一个很酷的窗口，也能真正运行 Python 3 量化投资分析平台。本书中的大部分示例代码都可以在 Python 3.6 和 Python 3.7 的平台上运行，包含 Windows 7/8/10 的 32 位或者 64 位、macOS、Linux 等操作系统，建议使用 Windows 64 位系统。

本书能够顺利完稿，首先要感谢电子工业出版社的黄爱萍和李淑丽两位编辑，在图书的编写过程中她们提出了很多宝贵的建议，还要感谢西安博成基金管理有限公司对本书在理念上的指导，同时，也非常感谢李宇昂对本书的排版布局及插图设计提供了很好的设计方案，使本书增色不少。

其次，感谢我的博客和公众号中的广大网友，这些朋友在我写书的过程中，给予了很大的支持，并对一些程序代码提出了许多值得改进的意见。

最后，感谢帮我检查并核对书稿的李美平、袁珊珊和房金思，谢谢你们！

接下来，我会一如既往地和诸位朋友一起完善这个开源量化平台，让我们一起加油吧！





電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

目 录

| | |
|------------------------------------|-----|
| 第 1 章 准备工作 | 1 |
| 1.1 Python 简介 | 1 |
| 1.2 Python 安装 | 3 |
| 1.3 Pip 包管理工具 | 13 |
| 1.4 Python 常用开发工具安装 | 19 |
| 1.5 Python 集成开发环境 Spyder 的使用 | 23 |
| 第 2 章 Python 的语法知识 | 28 |
| 2.1 Python 语言与其他语言对比 | 28 |
| 2.2 Python 编程基础 | 31 |
| 2.3 Python 的赋值语句 | 35 |
| 2.4 Python 的输入语句和输出语句 | 40 |
| 2.5 Python 程序流程控制语句 | 44 |
| 2.6 import 语句 | 51 |
| 第 3 章 Python 的数据与运算 | 59 |
| 3.1 Python 的数据类型 | 59 |
| 3.2 运算符及优先级 | 70 |
| 3.3 数值运算 | 83 |
| 3.4 字符串及相关操作 | 91 |
| 3.5 列表及相关操作 | 102 |



| | |
|-------------------------|-----|
| 3.6 集合及相关操作 | 110 |
| 第 4 章 自定义函数、类和作用域 | 120 |
| 4.1 Python 的自定义函数 | 120 |
| 4.2 Python 的类 | 132 |
| 4.3 Python 的作用域 | 146 |
| 第 5 章 NumPy 库与多维数组 | 153 |
| 5.1 NumPy 的简介 | 153 |
| 5.2 NumPy 库的安装和使用 | 154 |
| 5.3 ndarray 数组元素的数据类型 | 158 |
| 5.4 ndarray 数组的索引、切片和转置 | 160 |
| 5.5 NumPy 通用函数 | 166 |
| 5.6 ndarray 数组文件的保存和读取 | 168 |
| 第 6 章 Pandas 库与数据处理 | 171 |
| 6.1 Pandas 安装和使用 | 171 |
| 6.2 Pandas 数据结构 | 172 |
| 6.3 股票数据使用 | 174 |
| 6.4 DataFrame 数据操作 | 179 |
| 6.5 DataFrame 无效值 | 193 |
| 6.6 DataFrame 索引操作 | 194 |
| 6.7 DataFrame 数据的追加与合并 | 196 |
| 6.8 DataFrame 数据的保存和读取 | 199 |
| 6.9 DataFrame 运算 | 206 |
| 6.10 DataFrame 数据画线 | 208 |
| 6.11 仿通达信大智慧公式指标 KDJ | 210 |
| 第 7 章 Matplotlib 模块 | 217 |
| 7.1 Matplotlib 的使用 | 217 |
| 7.2 有关 Pyplot 显示的方法 | 233 |
| 7.3 Pyplot 常用绘图方法 | 236 |

| | |
|---------------------------|------------|
| 7.4 共享 x 坐标轴画图 | 239 |
| 7.5 绘制 K 线图 | 241 |
| 第 8 章 Tkinter 模块 | 245 |
| 8.1 Tkinter 的使用 | 245 |
| 8.2 Tkinter 控件的属性 | 250 |
| 8.3 Tkinter 主窗口 | 260 |
| 8.4 Toplevel 顶层子窗口 | 263 |
| 8.5 创建窗口菜单条 | 264 |
| 8.6 创建弹出菜单 | 266 |
| 8.7 控件的几何布局管理方法 | 269 |
| 8.8 Tkinter 常用控件 | 274 |
| 8.9 Tkinter 的事件和绑定 | 299 |
| 8.10 Ttk 控件 | 304 |
| 8.11 Tix 控件 | 312 |
| 第 9 章 小白量化投资分析平台 | 327 |
| 9.1 平台整体功能的划分 | 327 |
| 9.2 全局变量 HP_global | 329 |
| 9.3 全局变量初始化 HP_set | 330 |
| 9.4 本地数据及格式处理 HP_data | 332 |
| 9.5 公式基础函数库 HP_formula | 336 |
| 9.6 窗口容器库 HP_view | 340 |
| 9.7 指标绘图库 HP_draw | 344 |
| 9.8 回测系统库 HP_sys | 355 |
| 9.9 智能聊天对话系统 HP_robot | 364 |
| 9.10 策略编辑器 HP_edit | 369 |
| 9.11 总体框架构建模块 HP_MainPage | 370 |
| 9.12 主程序模块 HP_main | 388 |

| | | |
|--------|------------|-----|
| 第 10 章 | 分析回测与预测 | 390 |
| 10.1 | 投资分析方法 | 390 |
| 10.2 | 选股 | 390 |
| 10.3 | 择时买入 | 396 |
| 10.4 | 持仓分析——卖点信号 | 406 |
| 10.5 | 操作策略 | 412 |
| 10.6 | 多只股票量化回测 | 416 |
| 10.7 | 深度学习预测股价 | 424 |
| 10.8 | 股票数据网络爬虫 | 428 |

1

第 1 章

准 备 工 作

1.1 Python简介

Python 是一种高层次的，结合了解释性、编译性、互动性和面向对象的脚本语言，其设计具有很强的可读性。它是一种解释型语言，这意味着开发过程中没有了编译这个环节；它是一种交互式语言，这意味着可以在 Python 提示符“>>>”处直接输入程序语句；它是一种面向对象语言，这意味着它是一种支持把客观事物封装成抽象的类的编程技术。

对于初级程序员而言，Python 是一种伟大的语言。它支持广泛的应用程序开发，从简单的文字处理到浏览器再到游戏，无所不能。

由于 Python 语言具有简洁性、易读性及可扩展性，因此在国外科学计算的研究机构中的使用日益增多，一些知名大学也已经采用 Python 来教授程序设计课程。例如，卡耐基梅隆大学的编程基础、麻省理工学院的计算机科学和编程导论。众多开源的科学计算软件包都提供了 Python 的调用接口，如著名的计算机视觉库 OpenCV、三维可视化库 VTK、医学图像处理库 ITK。而 Python 专用的科学计算扩展库就更多了，如 NumPy, SciPy 和 Matplotlib 三个经典的科学计算扩展库，它们分别为 Python 提供了快速数组处理、数值运算及绘图功能。因此，由 Python 语言及其众多的扩展库所构成的开发环境十分适合工程技术和科研人员处理实验数据、制作图表，甚至



开发科学计算应用程序。

1. Python 的发展历史

Python 是由吉多·范罗苏姆 (Guido van Rossum) 于 20 世纪 90 年代初在荷兰国家数学和计算机科学研究所设计出来的。Python 本身也是由诸多其他语言发展而来的, 它自研发之初就遵循 GPL (General Public License) 协议 (一种开源协议)。这也是其迅速发展的主要原因之一。目前, Python 由一个核心开发团队维护, 而 Guido van Rossum 主要指导其进展, 仍然起着至关重要的作用。

Python 有两个最新版本, 即 Python 2.7.16 和 Python 3.7.3。Python 2 和 Python 3 是两个不同语法的版本, 很多旧库只支持 Python 2, 目前仍有小部分人在使用。Python 3 具有很多高级语言的特性, 功能更加强大。Python 2 对中文支持不是很好, 不支持中文文件名, 图形中也不支持中文显示。Python 2.7 将在 2020 年停止维护, 很多公司已经明确不再为 Python 2 提供后续版本库的支持, 因此, 本书主要介绍 Python 3.7 的开发编程。

Python 2 和 Python 3 语法不同的例子:

Python 2: `print 'Hello Word !'`

Python 3: `print('Hello Word !')`

2. Python 的特点

(1) 易于学习: 从小学生到 80 岁的老人, 只要认识英文字母, 就可以学习 Python 语言。浙江省对信息技术课程进行了改革, 从 2018 年开始将 Visual Basic 语言更换为 Python 语言, 甚至还将 Python 语言纳入了浙江省信息技术高考内容。

(2) 易于阅读: Python 代码定义得更加清晰, 其编写就像读英语一样简单。

(3) 易于维护: Python 成功的原因之一在于它的源代码相当容易维护。

(4) 一个广泛的标准库: Python 最大的优势之一是拥有丰富的、跨平台的资源库, 并且与 UNIX, Windows 和 Macintosh 的兼容性都很好。

(5) 互动模式: 互动模式的支持, 即通过 Python 系统终端输入代码, 按回车键就能获得代码运行结果。

(6) 可移植: 基于其开放源代码的特性, Python 已经被移植 (也就是使其工作) 到许多操作平台, 如 Windows, macOS, Linux, 甚至是安卓手机上。

(7) 可扩展: 即可以用 C 语言或 C++ 语言编写 Python 的模块。当你需要一段运

行很快的关键代码,或者想要编写一些不愿开放的算法时,就可以使用 C 语言或 C++ 语言完成这部分程序,然后从 Python 程序中调用。

(8) 数据库: Python 提供所有主流的商业数据库的接口。

(9) 图形用户界面 (Graphical User Interface, GUI) 编程: Python 支持 GUI 编程,拥有大量 GUI 开发框架可供选择。此外, Tkinter, Ttk, Tix 等图形扩展模块已经成为 Python 系统中的标准库。

(10) 可嵌入: 可以将 Python 嵌入到 C/C++ 程序中,让你的程序用户获得“脚本化”的能力。

1.2 Python 安装

1.2.1 安装 Python 3

进入 Python 官网的首页,如图 1-1 所示。

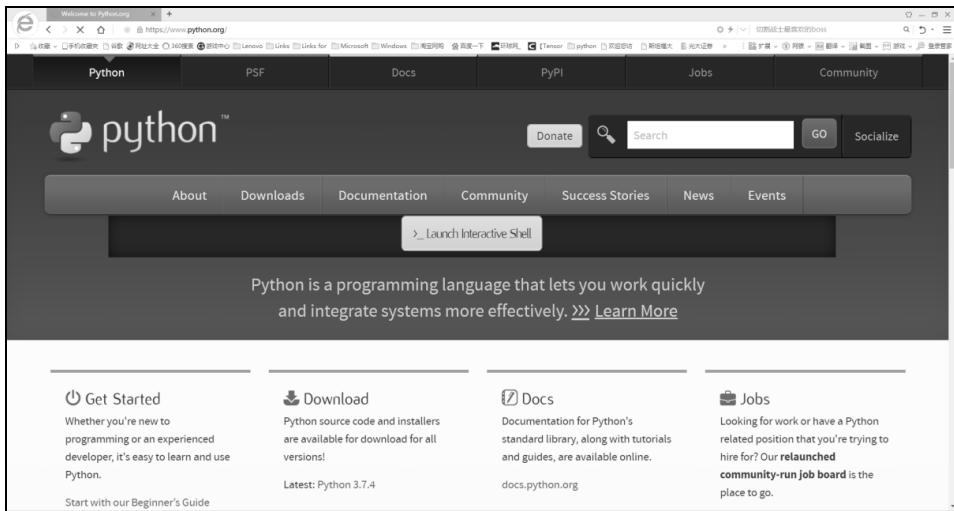


图 1-1 Python 官网页面

点击“Downloads”菜单,选择“All release”选项,进入如图 1-2 所示的页面。

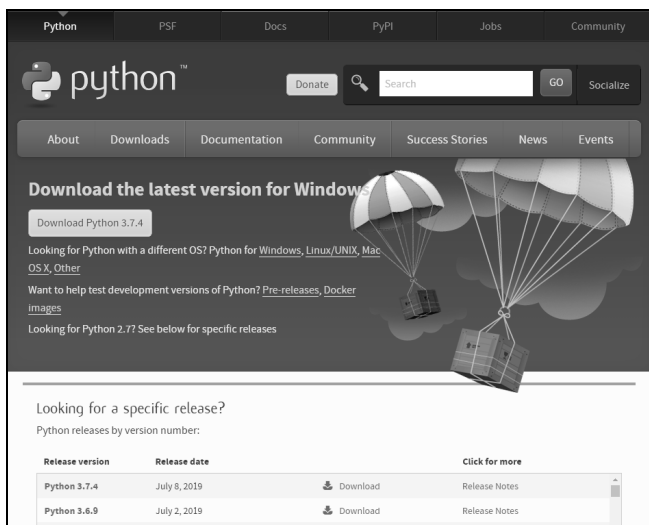


图 1-2 Python 下载页面

我们可以看到，Python 3 除了 Windows 版本外，还有 Linux/UNIX，macOS 及其他版本。Python 版本又分 32 位和 64 位系统。

如果想选择其他版本号的软件，可以翻看下面的不同版本号，点击后面的“Download”按钮即可下载。

我们选择 Python 3.7.1 的 64 位版本，点击“Windows x86-64 executable installer”开始下载文件“python-3.7.1-amd64.exe”。下载完成后，双击文件图标，自动运行安装程序，如图 1-3 所示。

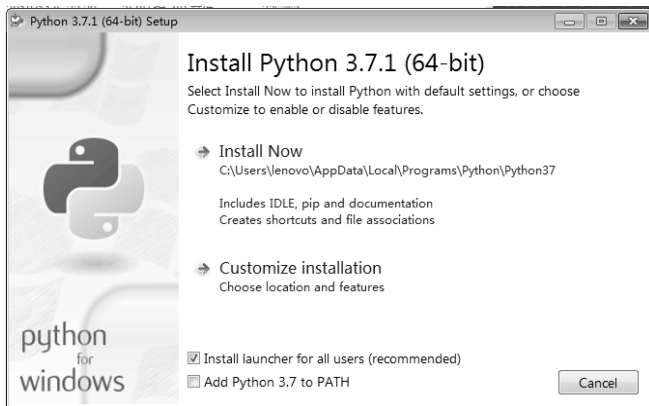


图 1-3 Python 3.7.1(64-bit)安装对话框

点击“Install Now”默认安装。安装成功后，出现如图 1-4 所示的对话框，点击“Close”按钮，安装结束。

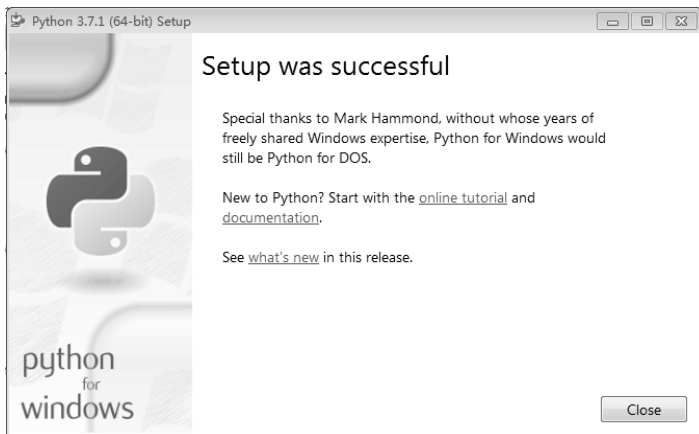


图 1-4 Python 安装成功对话框

在 Windows 程序栏目中就会出现“Python 3.7 (64-bit)”，如图 1-5 所示。点击后，开始运行 Python 程序，如图 1-6 所示。

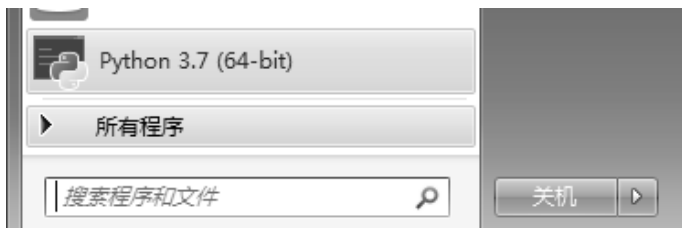


图 1-5 Windows 程序栏目

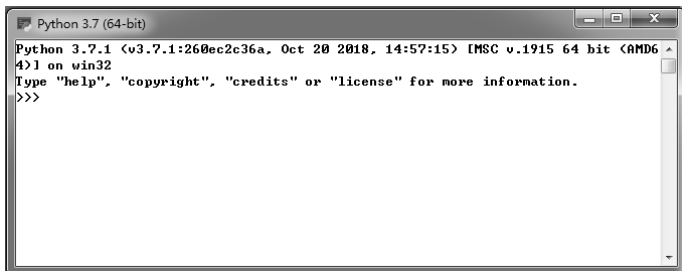


图 1-6 Python 3.7 运行窗口

此时, Python 3.7 只是基础运行环境, 很多 Python 开发包如 NumPy, Pandas 等都还没有, 需要安装。而如果通过 Anaconda 软件来安装 Python 开发环境就可以解决 99% 的包导入的问题。

1.2.2 通过 Anaconda 安装 Python

Anaconda 是一个和 Canopy 类似的科学计算环境, 但其用起来更加方便, 自带的包管理器 conda 很强大。

Anaconda 提供了 Python 3.7 和 Python 2.7 两个主要的版本, 如果需要其他版本可以通过 conda 来创建。

Anaconda 还提供了 Spyder, IPython 等 Python 开发工具环境。

Anaconda 也支持不同操作系统, 如 Windows, macOS, Linux 等操作系统。

输入 Anaconda 的下载网址就可以进入如图 1-7 所示的下载页面。

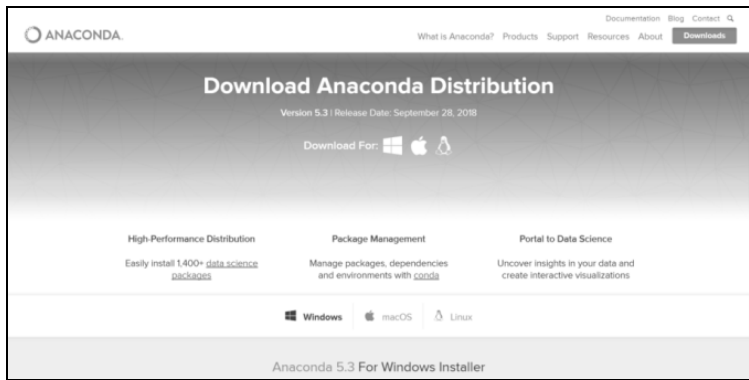


图 1-7 Anaconda 的下载页面

我们可以根据自己使用的操作系统, 选择对应的 Anaconda 安装软件来下载。

不同的操作系统的安装过程类似, 下面我们分别给大家介绍在 Windows 系统、macOS 系统和 Linux 系统下, Anaconda 的安装过程。

1. Windows 系统安装 Anaconda

在图 1-7 所示的下载页面中, 选择 Windows 系统则会出现要安装的 Python 版本和所对应的 64 位或 32 位系统安装程序, 如图 1-8 所示。

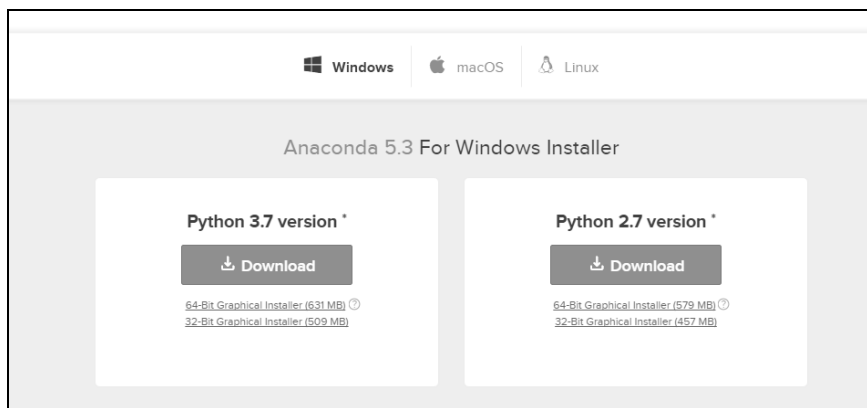


图 1-8 Anaconda 所对应的 Python 版本号

下载 Python 3.7 版本，有两个安装包。即

64 位图形安装包：64-Bit Graphical Installer (631 MB)。

32 位图形安装包：32-Bit Graphical Installer (509 MB)。

我们选择“64-Bit Graphical Installer (631 MB)”进行安装。开始安装后，出现如图 1-9 所示的对话框。

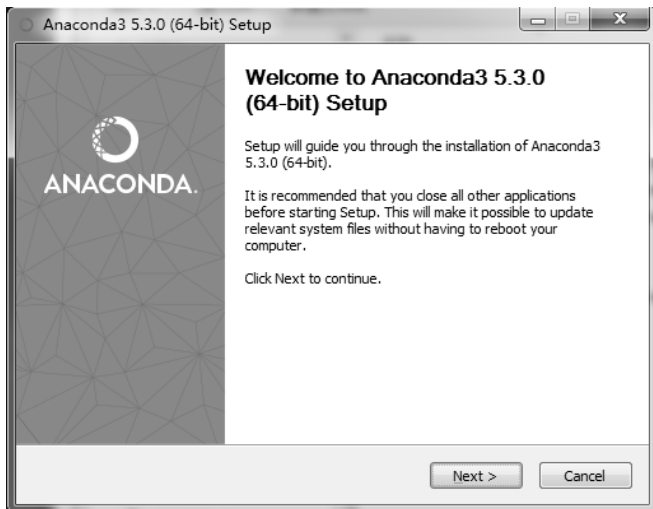


图 1-9 Anaconda3 5.3.0 安装程序对话框

我们一直选择“Next>”按钮，采用默认方式安装。安装结束后，会出现如图 1-10 所示的对话框。

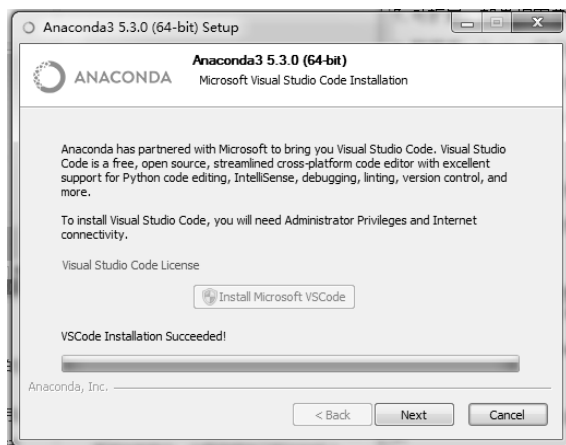


图 1-10 Anaconda 安装结束对话框

在结束对话框中选择“Next”按钮，安装程序完成并退出。

我们可以在 Windows 的程序栏中，点击“Anaconda”来运行 Anaconda 软件。

在 Anaconda 软件安装的过程中，有个细节一定要注意。在出现图 1-11 的窗口时，要选中图中的 1、2 复选框来注册运行环境，否则 Anaconda 软件安装成功后无法直接使用。

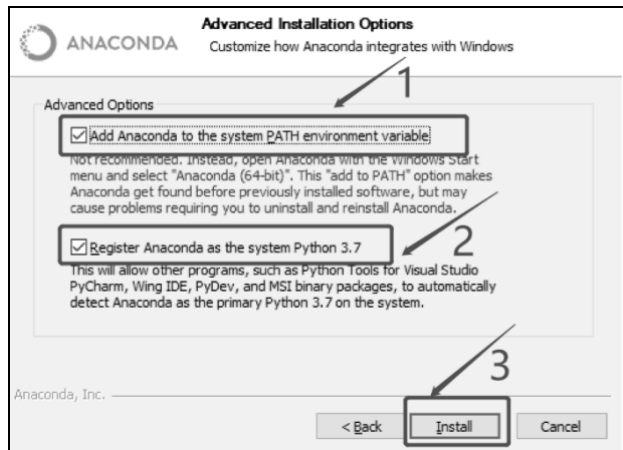


图 1-11 Anaconda 软件安装窗口

2. macOS 系统安装 Anaconda

在此系统下的 Python 也分为 Python 3.7 和 Python 2.7 两个版本。在 macOS 系统

下没有 32 位版本，只有两种安装方式，即图形安装方式和命令行安装方式，如图 1-12 所示。

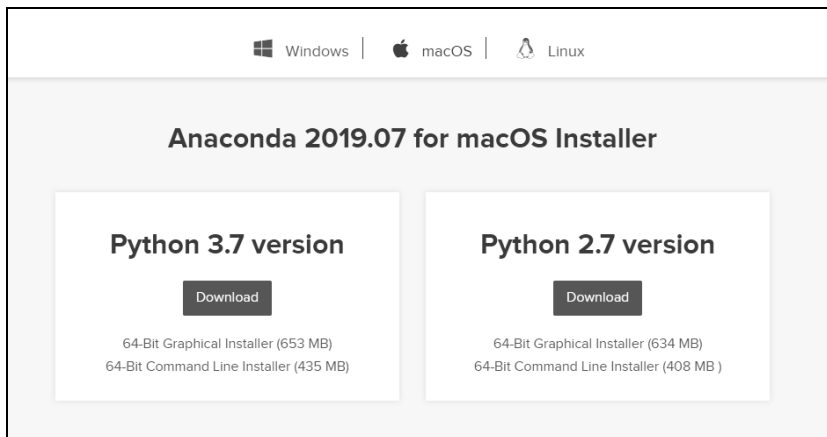


图 1-12 Anaconda 的 macOS 安装页面

Python 3.7 的安装文件有以下两个。

图形安装文件：64-Bit Graphical Installer (653 MB)。

命令行安装文件：64-Bit Command-Line Installer (435 MB)。

1) 图形安装方式

在下载目录中双击文件 “Anaconda3-5.3.0-macOS-x86_64.pkg”，开始执行安装程序。如图 1-13 所示。



图 1-13 图形安装对话框

在安装过程中，macOS 系统安装 Anaconda3 软件需要先阅读安装信息，如图 1-14 所示。阅读完成后，点击“继续”按钮才能继续安装。

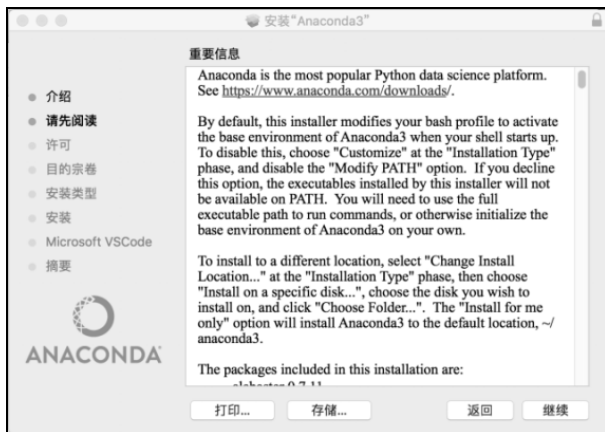


图 1-14 阅读 Anaconda 安装信息

此时弹出小窗口，如图 1-15 所示。我们必须选择“同意”按钮才能继续安装，如果选择“不同意”就会退出安装程序。



图 1-15 安装软件许可协议

在同意安装软件许可协议后，软件会继续安装，此时提示选择软件安装位置，如图 1-16 所示。

点击“安装”按钮，选择默认位置安装。安装成功后会出现“安装成功”窗口，如图 1-17 所示。

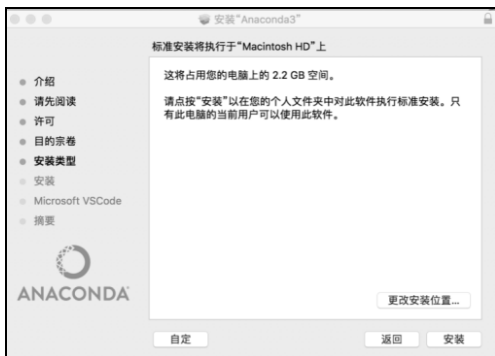


图 1-16 选择安装位置

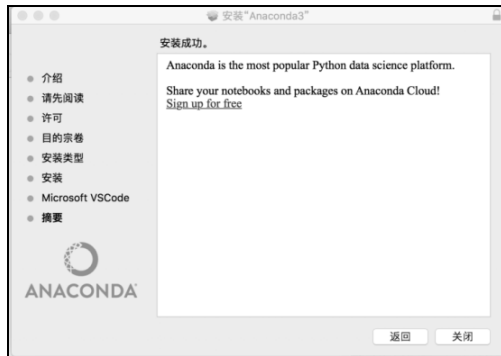


图 1-17 “安装成功”窗口

关闭安装程序，打开启动台会看到 Anaconda 的软件图标，如图 1-18 所示。



图 1-18 macOS 启动台

双击“Anaconda-Navigator”图标启动软件，进入 Anaconda 程序主画面，如图 1-19 所示。

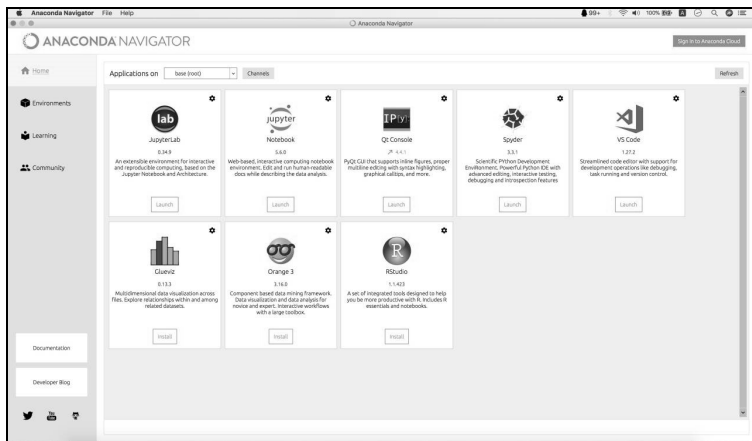


图 1-19 Anaconda 软件启动窗口

2) 命令行安装方式

首先下载安装文件“64-Bit Command-Line Installer (544 MB)”，然后在苹果系统终端上切换到“Downloads”目录，输入下面命令，开始安装 Anaconda，如图 1-20 所示。

```
sh Anaconda3-5.3.0-Mac OSX-x86_64.sh
```

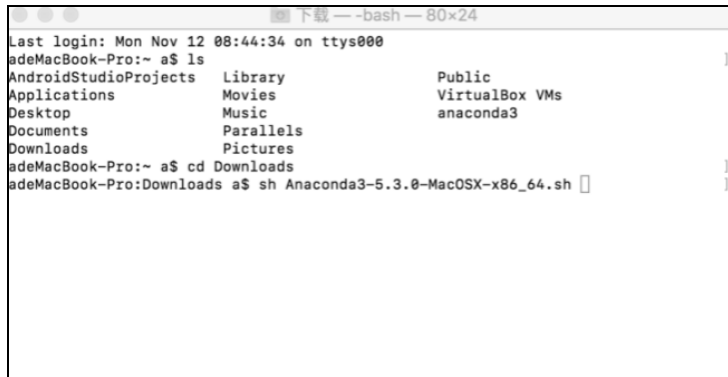


图 1-20 macOS 系统终端

3. Linux 系统安装 Anaconda

Linux 版本的 Anaconda3 安装程序有三个版本。

- (1) 64-Bit (x86) Installer (652.5 MB)。
- (2) 64-Bit (Power8 and Power9) Installer (313.6 MB)。
- (3) 32-Bit Installer (542.7 MB)。

我们选择“64-Bit (x86) Installer (652.5 MB)”版本下载，下载文件为“Anaconda3-2018.12-Linux-x86_64.sh”。下载完成后，就可以打开终端进行安装了。

我们以 Ubuntu 16 系统中文桌面版为例，下载目录为中文“下载”。

打开终端，在命令符号“\$”后输入以下命令。

```
cd 下载
sudo sh Anaconda3-2018.12-Linux-x86_64.sh -u
```

然后会出现注册信息，一直按回车键。在阅读完注册信息并提示“yes/no”时，通常都选“yes”来默认安装。

安装成功后，在终端输入以下命令启动 Anaconda3。

```
source ~/anaconda3/bin/activate root
anaconda-navigator
```

1.2.3 下载安装绿色 Python 3.7 版本

从我们提供的资源包中把绿色 Python 下载到任意盘符中，解压后，点击文件“Py37.bat”运行。

我们提供的绿色版本 Python 3.7（简称绿色 Py37）除了集成了 Windows 64 位系统下的 Python 3.7、Spyder 开发工具和丰富的科学计算包外，还集成了 QUANTAXIS 数据包、Tushare 财经数据接口包、聚宽数据 JQData 股票数据包（聚宽数据 JQData 需要在官网注册用户）、OpenDataTools 股票数据包等，用户可以直接使用这些免费的金融数据。而如果用其他方法安装的 Python 环境，则需要用 pip 命令来安装这些金融数据包。因此，建议读者先下载使用绿色 Py37。

1.3 Pip包管理工具

Python 的库是由众多包和模块组成的，其中能够完成某类完整功能所必需的基开发包是由众多功能丰富的类和函数组成的。举例来说，NumPy 库就是一个具有强大科学计算功能的函数库。NumPy 库不是 Python 的一部分，是第三方提供的，因此只有安装了 NumPy 库才能使用其中的功能和函数。Python 库的文件一般都用 pip 软件来安装管理。

pip 是一个 Python 包管理工具，主要用于安装 PyPI（Python Package Index）上的软件包，它可以替代旧的 easy_install 工具。PyPI 是 Python 官方的第三方库的仓库，所有人都可以使用 pip 包管理器下载第三方库或上传自己开发的库到 PyPI。Pip 可正常工作在 Windows，macOS，UNIX/Linux 等操作系统上。

1. pip 软件安装

pip 是 Python 官方推荐的包管理工具。

如果你是使用 Anaconda 安装的 Python，或者使用我们提供的绿色版本 Py37，则里面已经包含了 pip 安装程序。

打开“Py37”目录，如“D:\py37”。单击鼠标右键弹出菜单，以管理员身份运行程序“WinPython Command Prompt.exe”就会出现“Command Prompt”命令窗口。这里可以直接输入 pip 安装命令来安装库文件。

例如：

```
pip install numpy
```

如果你是自己下载安装的 Python 3.7，就需要下载安装 pip 工具软件，方法如下。pip 下载网址和页面如图 1-21 所示。

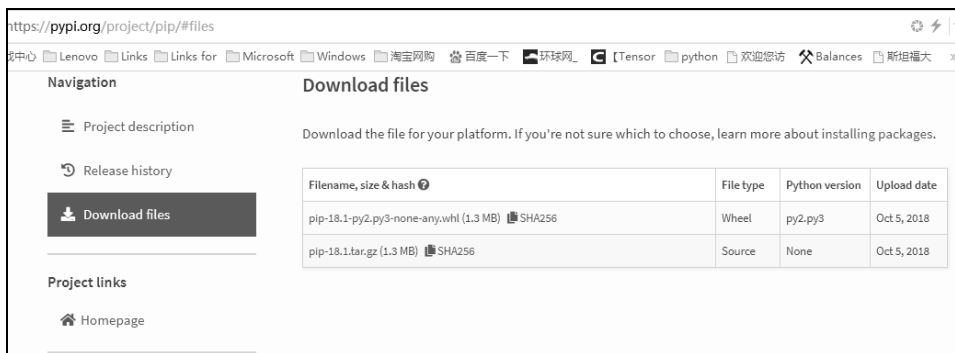


图 1-21 pip 下载网址和页面

下载“pip-18.1.tar.gz (1.3 MB)”压缩包，解压后，在目录中进行安装。

pip 软件的安装方式如下：

(1) 在 Windows 下执行如下命令：

```
Python setup.py install
```

(2) 在 Ubuntu Linux 下执行如下命令：

```
sudo apt-get install python-pip
```

安装成功后，可以执行以下命令查看已经安装好的包名。

```
pip list
```

2. pip 软件升级

(1) 在 Windows 下执行如下命令：

```
python -m pip install -U pip
```

(2) 在 Linux 或 macOS 下执行如下命令:

```
pip install -U pip
```

3. 新库的安装方法

(1) 在 Windows 下利用“Win + R”组合键打开运行窗口, 输入“cmd”回车, 找到 pip 命令所在路径, 在命令行中切换至该目录。

```
cd C:\Python37\Scripts
```

输入以下命令:

```
pip install <库名称>
```

例如:

```
pip install numpy
```

(2) 下载 whl 文件也可以安装库。

whl 文件是 Python 扩展包, 包含了 py 文件, 以及经过编译的 pyd 文件。下载相应库的 whl 文件也可以安装库, 但要注意, 要下载相应版本的并选择对应的 32 位或 64 位系统。

在 cmd 上输入下面命令即可。

```
pip install xxx.whl
```

4. 量化分析所需的库

下面是小白量化投资软件需要安装的 Python 库或模块及其功能。

(1) NumPy 是基于 Python 科学计算的第三方库, 提供了矩阵、线性代数、傅立叶变换等解决方案。

(2) SciPy 是基于 Python MATLAB 实现的库, 旨在实现 MATLAB 的所有功能。

(3) Matplotlib 是利用 Python 实现类 MATLAB 的第三方库, 主要用于绘制一些高质量的数学二维图形。绘制 K 线还需要安装 mpl_finance 包。

(4) PILlow 是基于 Python 的图像处理库, 功能强大, 对图形文件的格式支持广泛。

(5) Pandas 是数据整理的完美工具。

(6) PyGame 是基于 Python 的多媒体开发和游戏软件开发的模块。

- (7) Jieba 为中文分词工具。
- (8) 金融数据接口库，如 tushare, jqdatasdk 等。
- (9) 网页获取信息及网络爬虫所需要的模块，如 requests, bs4。
- (10) 机器学习需要安装 sklearn, keras 等模块。

5. 卸载库

卸载库用以下命令：

```
pip uninstall <库名称>
```

例如：

```
pip uninstall numpy
```

6. 在 Anaconda3 中安装库

在 Windows 的程序中，选择 “Anaconda3 (64-bit)” 下面的 “Anaconda Prompt” 程序。如图 1-22 所示。

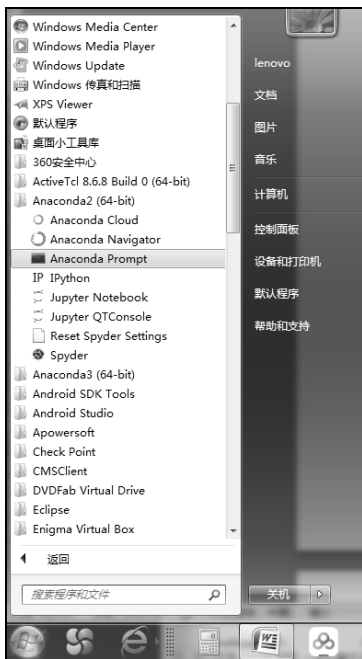


图 1-22 选择 Anaconda Prompt

程序运行后会出现“Anaconda Prompt”窗口，然后输入 pip 安装命令来安装文件即可，如图 1-23 所示。

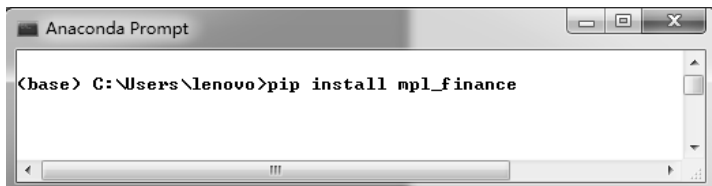


图 1-23 输入 pip 安装命令

7. 量化数据支持包

在金融量化投资领域中，进行数据分析就是利用海量数据信息对金融经济活动进行预测，并根据历史预测情况及时反馈预测效果并更新投资策略动态，以实现最佳的预测效果。例如，在股票量化分析中，必须要有历史行情数据。下面我们介绍一些能够提供股票行情数据等金融数据的平台及支持 Python 的数据包。

1) 聚宽数据（JQData）数据包

JQData 是聚宽数据团队专门为有志于从事量化投资的金融机构、研究人员及个人量化爱好者提供的本地量化金融数据。用户只需要在本地 Python 环境下安装 JQData 数据包，输入三行代码，即可调用由聚宽数据团队专业生产的全套量化金融数据，这可以让你轻松告别平台限制，灵活安全地完成本地化的量化研究与投资决策。

支持系统：Linux，macOS，Windows。

支持 Python 2 和 Python 3。

在聚宽官网上可以申请 JQData 用户。

JQData 自动安装方法如下：

进入 Python 所在目录，执行以下代码安装。

```
pip install git+https://github.com/JoinQuant/jqdatasdk.git
```

或利用以下安装命令快速安装。

```
pip install -U git+https://github.com/JoinQuant/jqdatasdk.git -i  
https://mirrors.aliyun.com/pypi/simple/
```

在 Linux 系统的 Anaconda3 中，使用如下命令安装 JQData 数据包。

```
pip install jqdatasdk --user
```

2) QUANTAXIS 量化金融策略框架

QUANTAXIS 量化金融策略框架具有数据爬取—清洗存储—分析回测—可视化—交易复盘的本地一站式解决方案。

QUANTAXIS 量化金融策略框架，是一个面向中小型策略团队的量化分析解决方案。它是一种通过高度解耦的模块化及标准化协议快速实现面向场景的定制化解决方案。QUANTAXIS 是一个渐进式的开放式框架，可以根据自己的需要引入数据、分析方案、可视化过程等，也可以通过 RESTful 接口快速实现多人局域网/广域网内的协作。

安装命令如下：

```
pip install quantaxis
```

3) Tushare 财经数据接口包

Tushare 是一个免费的、开源的 Python 财经数据接口包。

安装命令：

```
pip install tushare
```

在 Linux 系统的 Anaconda3 中，使用如下安装命令：

```
pip install tushare --user
```

4) OpenDataTools 免费财经数据

由于 OpenDataTools 财务数据只支持 Python3，因此要安装 Python 3.6 以上版本。

安装命令如下：

```
pip install opendatatools
```

在 Linux 系统的 Anaconda3 中，使用如下安装命令：

```
pip install opendatatools --user
```

本节介绍了 Python 库的安装和管理操作。而在 Python 程序中，要先通过 import 语句引入对应的库，才能使用它们中的功能和函数。

1.4 Python常用开发工具安装

我们可以用 Python 自带的简易开发工具 IDLE 编写 Python 代码，也可以用记事本来写 Python 代码，但是这些都不是很直观和方便。因此，很多 Python 开发者就会使用 Spyder 和 PyCharm 编写代码。

在绿色 Py37 压缩包中已经包含 Spyder 开发工具，执行批处理文件“Py37.bat”就会启动 Spyder 软件。如果你下载了 Anaconda，其中就有 Spyder 软件，双击“Spyder”栏目就能启动。

我们建议初学者先使用 Spyder 软件。

1. Spyder 安装

如果没有安装过 Spyder，就可以在 Spyder 的官网下载。根据自己的需要选择合适的版本号进行安装。见图 1-24。

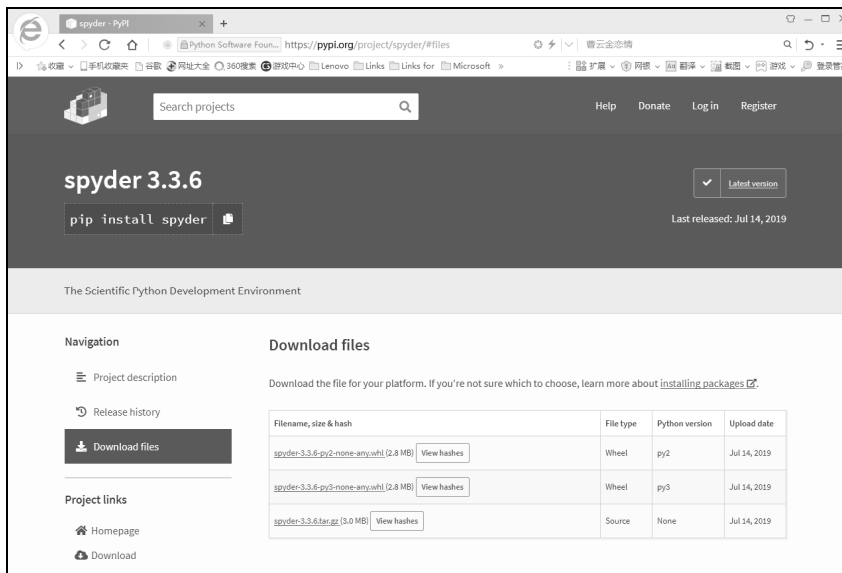


图 1-24 spyder 3.3.6 的下载页面

我们选择下载“spyder-3.3.6-py3-none-any.whl”文件，下载完成后，在 Windows 的 cmd 窗口切换到该 whl 文件目录下，输入如下命令进行安装。

```
pip install spyder-3.3.6-py3-none-any.whl
```

图 1-25 是 Spyder 启动后的画面。

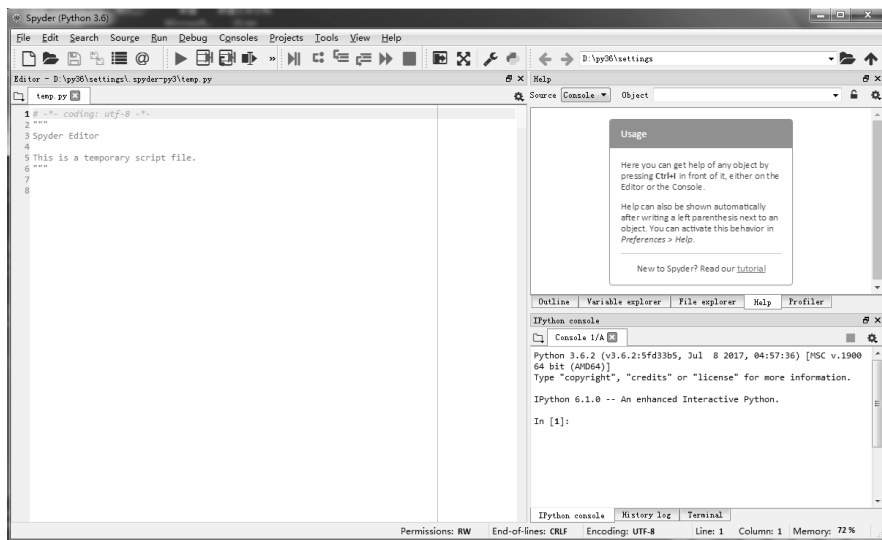


图 1-25 Spyder 启动后的窗口

2. PyCharm 安装

我们可以在 PyCharm 的官网下载，其页面如图 1-26 所示。

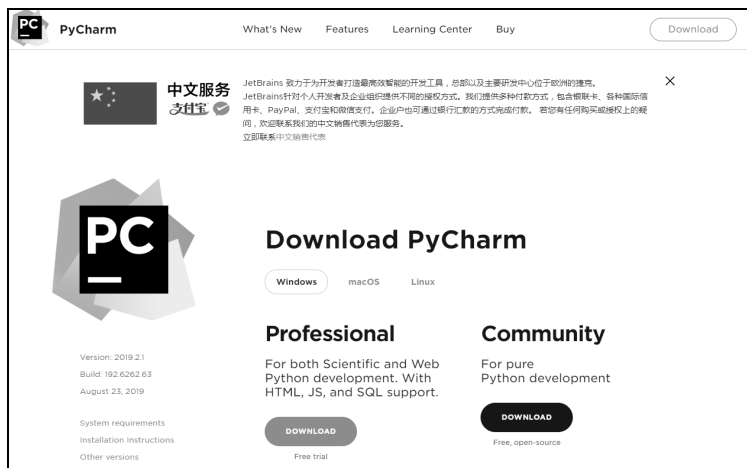


图 1-26 PyCharm 官网页面

我们以下载 Community 社区版为例。下载完成后，进行 PyCharm 软件安装，如图 1-27 所示。



图 1-27 PyCharm 软件安装对话框

选择 PyCharm 软件所要安装的目录，如图 1-28 所示。

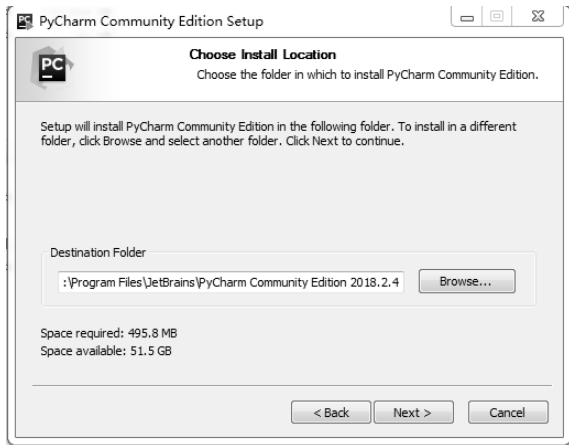


图 1-28 PyCharm 软件安装目录

我们也可以不选择目录，直接点击“Next>”按钮，选择默认目录安装。然后选择要安装的软件类型，32 位或 64 位，如图 1-29 所示。这里我们选择 64 位。

然后点击“Next>”按钮，软件开始自动安装，安装结束出现如图 1-30 所示的对话框。

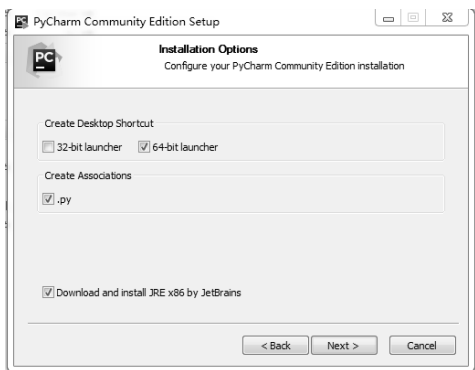


图 1-29 选择软件类型



图 1-30 PyCharm 软件安装完成对话框

点击“Finish”按钮，安装程序结束并退出。运行 PyCharm 软件，出现图 1-31 所示的画面。



图 1-31 PyCharm 软件启动画面

3. 在 Windows 的 PyCharm 新项目中使用绿色 Py37 的环境

由于 PyCharm 自带的 Python 环境缺少很多必要的包，因此我们可以在创建新工程文件时选择绿色 Py37 的 Python 环境或者 Anaconda 的 Python 环境，因为它们已经集成了很多常用工具包。

点击“Create New Project”，出现如图 1-32 所示的对话框。

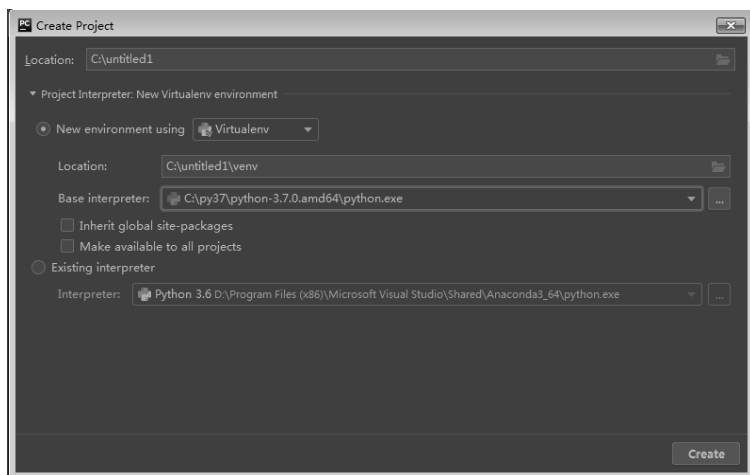


图 1-32 选择 Python 环境

图 1-32 中的 Location 是选择创建 Python 工程的位置及工程名字（根据自己的情况选择，默认为 C 盘），笔者的工程目录为 C:\untitled1，工程名字为 untitled1（可以随便取）。图中的“Base Interpreter”是安装的 Python 解释器，可以选择默认，也可以自己修改。这里我们修改为 D:\py37\python-3.7.0.amd64\python.exe，然后点击“Create”按钮。

如果已经安装了 Anaconda3，则可以选择“C:\ProgramData\Anaconda3\python.exe”。

1.5 Python集成开发环境Spyder的使用

由于 Spyder 是小白量化投资系统框架的主要开发工具，所以我们着重介绍一下。

Spyder 是 Python(x,y)的作者开发的一个简单的集成开发环境。和其他的 Python 开发环境相比，其最大的优点就是具有模仿 MATLAB 的“工作空间”的功能，这可以很方便地观察和修改变量的值。

1. Spyder 操作与设置

Spyder 的界面由许多窗格构成，用户可以根据自己的喜好调整它们的位置和大小。当多个窗格出现在同一个区域时，就会使用标签页的形式显示。例如，在图 1-33

中，可以看到 Editor，Object inspector，Variable explorer，File explorer，Console，History log，以及两个显示图像的窗格。在“View”菜单中，可以设置是否显示这些窗格。

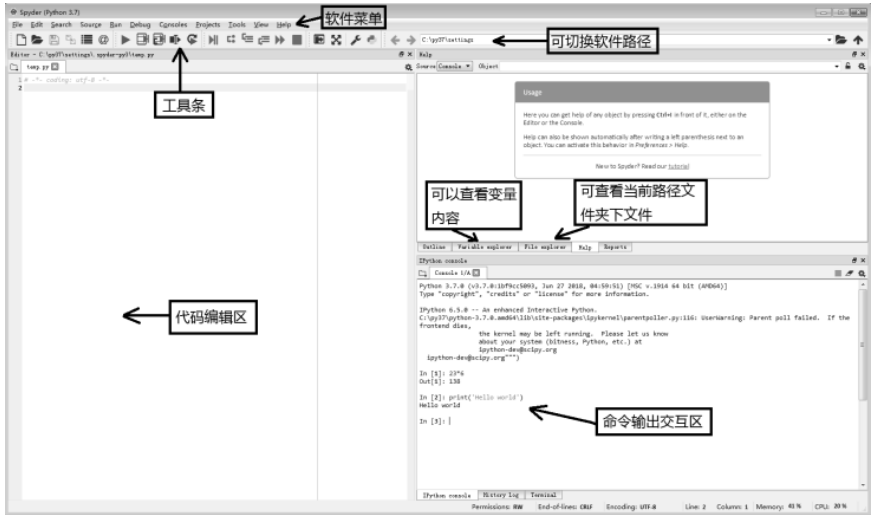


图 1-33 Spyder 软件多窗口显示

我们可以看到，Spyder 的界面设计和 MATLAB 十分相似，因此熟悉 MATLAB 的读者可以很快习惯使用 Spyder。

2. 常用快捷键

Spyder 常用快捷键如表 1-1 所示。

表 1-1 Spyder 常用快捷键

| 快捷键 | 功能 |
|-----------|----------------------------------|
| F5 | 运行当前编辑器中的程序 |
| Ctrl+I | 注释/取消注释代码 |
| Ctrl+I | 清空 Console 信息 |
| Ctrl+c | 中止程序运行 |
| Shift+Tab | 调整代码的缩进 |
| Ctrl+F5 | 进入代码，调试 debug |
| Ctrl+F10 | 单步调试执行语句 |
| Clear | 在控制台 Console 中输入 Clear 命令，清除所有命令 |
| Reset | 在控制台 Console 中输入 Reset 命令，清除所有变量 |

3. 设置默认目录

设置默认工作路径的方式为从菜单“Tools”中选择“Preferences”，再选择“Current working directory”，就会出现如图 1-34 所示的设置默认工作路径窗口，可以修改路径，如“d:\tt”目录。

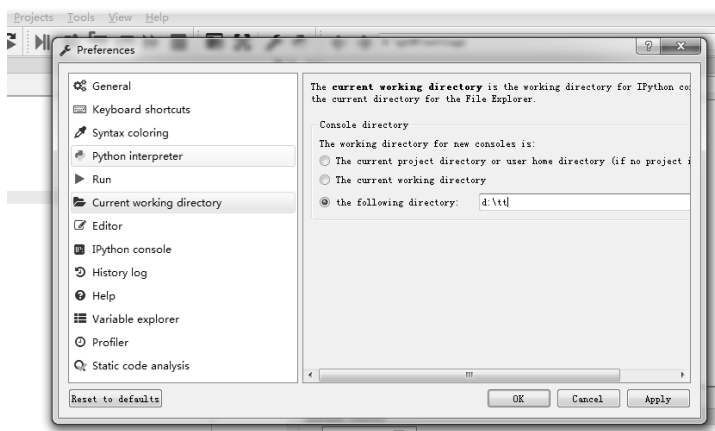


图 1-34 设置默认工作路径

4. 设置界面风格及字体

1) 界面风格设置

我们可以从“View”的下拉菜单中选择“Window layouts”，再选择“Spyder Default Layout”，如图 1-35 所示。

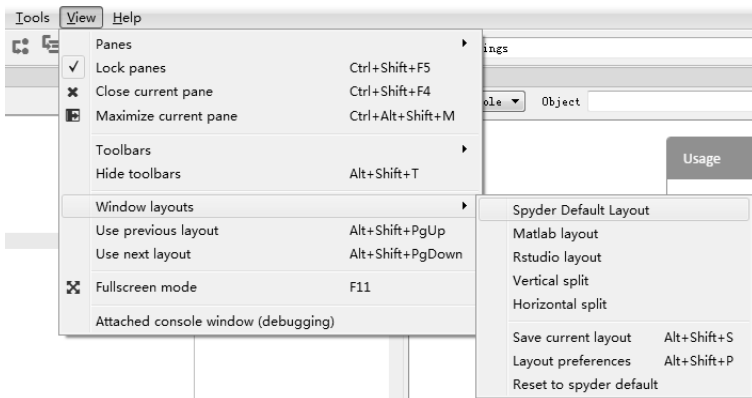


图 1-35 界面风格设置

用户可以根据自己的喜好选择界面,如熟悉使用 MATLAB 的用户可以选择将界面设置为“Matlab layout”风格。

2) 字体设置

可以从菜单“Tools”中选择“Preferences”栏目,如图 1-36 所示,然后再选择“Syntax coloring”字体颜色。

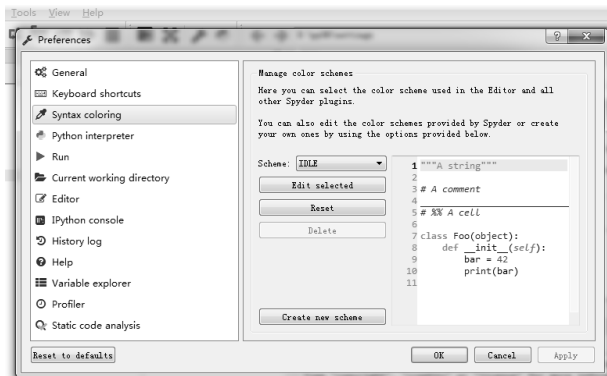


图 1-36 选择“Syntax coloring”字体颜色

在“General”栏目中能设置字体和字体尺寸,如图 1-37 所示。

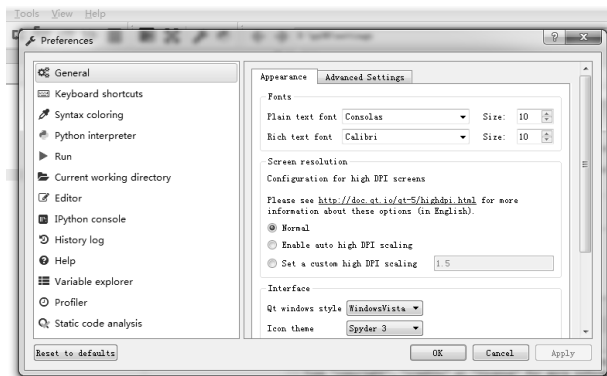


图 1-37 设置字体和字体尺寸

5. 代码运行与调试

- (1) 按“F5”键将运行当前编辑器中的程序。
- (2) 按“Ctrl+F5”组合键进入代码调试。

在调试符“ipdb>”下输入“c”命令或者按“Ctrl+F12”组合键可以让程序执行到下一个断点，“q”命令则是退出调试。

(3) 双击行首设置断点，按住“Ctrl+Shift”组合键，然后双击行首可以设置条件断点。

(4) 按“F9”键将运行当前编辑器中选中的程序代码行。

除了键盘的快捷键外，还可以用鼠标点击工具条中的图标来运行或调试程序。如图 1-38 所示。



图 1-38 Spyder 工具条的功能

2

第 2 章

Python 的语法知识

2.1 Python语言与其他语言对比

如果没有特殊强调，则本书主要讲的是 Python 3 的语法结构。

如果你懂一些其他计算机编程语言，如 java, C++, Visual FoxPro 等，那么通过与它们的对比，也可以帮助你快速了解 Python 3 编程语言。

Python 3 是脚本语言，也称为“胶水语言”，其程序简洁清晰，不需要编译就能直接运行。

表 2-1 所示为 Python 3 语言与其他语言的对比，不同的语言都有特色语句，表中给出了部分相同或相似功能的程序语句或命令。

表 2-1 Python 语言与其他语言的语法对比

| 语法命令 | Python 3 | Java | FoxPro | C++ |
|--------|---------------------------------|-----------|-------------------------------|-----------|
| 命令大小写 | 区分 | 区分 | 不区分 | 区分 |
| 标识符大小写 | 区分 | 区分 | 不区分 | 区分 |
| 单行注释 | # | // | * | // |
| 多行注释 | 三个单引号括起来"..." 或三个双引号括起来"..." | /* ... */ | text noshow ... endtext | /* ... */ |
| 命令行末注释 | # | // | && | // |



续表

| 语法命令 | Python 3 | Java | FoxPro | C++ |
|----------|---|---|---|---|
| 变量类型 | 赋值决定; 变量类型可变 | 类型定义; 变量类型不可变 | 赋值决定; 变量类型可变 | 类型定义; 变量类型不可变 |
| 变量模式 | 内置数据类型和对象数据类型; 字符变量是对象数据类型 | 内置数据类型和对象数据类型; 如 String 是对象数据类型 | 内置数据类型 | 内置数据类型, 映射 内存地址 |
| 命令块 | 以空格数决定 | { } | END 字母开头的命令结束 | { } |
| 命令续行 | \ | 无论多少行, 分号 “;” 前都是一条命令 | ; | \ |
| If 命令 | if <条件>: 语句 elif <条件>: 语句 else : 语句 例子: if num > 0: print("正数\n") elif num == 0: print("零\n") else: print("负数\n") | if <条件> { 语句; } else if <条件> { 语句; } else { 语句; } 例子: if (num > 0) { System.out.println("正数"); } else if (num == 0) { System.out.println("零"); } else { System.out.println("负数"); } | if<条件> 语句 else if<条件> 语句 else 语句 endif endif 例子: if num > 0 print "正数" else if num == 0 print "零" else print "负数" endif endif | if <条件> { 语句; } else if <条件> { 语句; } else { 语句; } 例子: if (num > 0) { printf("正数\n"); } else if (num == 0) { printf("零\n"); } else { printf("负数\n"); } |
| for 循环 | for i in range(10) : print(i==str(i)) | for(int i=0;i<10;i++) { System.out.println("i="+i); } } | for i=0 to 9 print 'i='+ str(i) endfor | for(int i=0;i<10;i++) { printf("i=%d",i); } } |
| while 循环 | while<条件> : 语句 else : 语句 | while <条件> { 语句; } } | do while <条件> 语句 end do | while <条件> { 语句; } } |

续表

| 语法命令 | Python 3 | Java | FoxPro | C++ |
|----------------------------------|--------------|--------------|--|---------------|
| break 中止循环 | break | break | break | break |
| continue 跳过当前循环的剩余语句，然后继续进行下一轮循环 | continue | continue | continue | continue |
| pass 空语句 | pass | { } | 空行 | { } |
| 命名空间 | 有 | 有 | 无 | 有 |
| 全局变量 | global | public | public | extern |
| 引入文件 | import <文件名> | import <文件名> | #include <文件名> set procedure to <文件名> | include <文件名> |

表 2-2 所示为 Python 语言与其他语言的数据类型对比。

表 2-2 Python 语言与其他语言的数据类型对比

| 数据类型 | Python 3 | Java | FoxPro | C++ |
|--------------------|---|-------------------------------------|--|--------------------------------------|
| 变量类型 | 赋值同时定义类型 | 事先定义 | 赋值同时定义类型 | 事先定义 |
| 布尔类型 逻辑变量 | 真 True，假 False | 真 True，假 False | 真.T.，假.F. | 真非 0 值，假 0 值 |
| 整数 int | 赋值同时定义类型 | byte, short, int, long | 赋值同时定义类型 | byte, short, int, long |
| 浮点数 | 赋值同时定义类型 | float/double | 赋值同时定义类型 | float/double |
| 字符串 | 赋值同时定义类型 | String | 赋值同时定义类型 | char[]或 char* |
| 数组/列表 (能够用下标访问) | 列表 list 使用方括号[], 赋值时定义，下标初值为 0。 例如: nums=[1, 3, 5, 7, 20] | 在定义类型时定义，下标初值为 0。 例如: int a[100] | dime, 预先定义，下标初值为 1。 例如: dime a[100] | 在定义类型时定义，下标初值为 0。 例如: char a[100] |
| 元组 | 元组与列表类似，不同之处在于元组的元素不能修改；元组使用小括号() | 无 | 无 | 无 |
| 字典 | 字典是无序的对象集合，字典由键和对应的值组成。 dict = {'姓名': '通通', 'QQ': '2775205'} | 无 | 无 | 无 |
| 其他数据类型 | 由 import 引入包带来的新数据类型。DataFrame 是 Pandas 的表格型数据结构 | 类似自定义类型: class | 类似自定义类型: class | 类似自定义类型: class, struct |
| 复数 complex | 有 | 无 | 无 | 无 |

2.2 Python编程基础

本节适合编程零基础的读者，如果已有 Python 3 基础，则可跳过。

为了让读者能够快速入手，本书提供了教学示例程序文件。当你有新想法时，可以将示例代码复制到新建文件中，修改后看看是否能得到想要的运行结果，这是一种比较好的学习方法。

1. Python 程序的行

- (1) 一般一条语句占有一行。
- (2) 模块之间用空行进行间隔。
- (3) 如果语句太长，就可以用斜线“\”进行跨行。

见下面的例子（示例 2-1 跨行演示.py）。

```
a=15
if a>=10 :
    print('a 大于等于 10! ')
else :
    print('a 小于 10! ')

print("好好学习，天天向上。我喜欢学 Python。")
#下条命令，用符号\分为 3 行。
print("好好学习， \
天天向上。 \
我喜欢学 Python。")
```

上面程序中，print 语句中使用换行符“\”把 1 行语句分成了 3 行。程序运行结果如下。

```
a 大于等于 10!
好好学习，天天向上。我喜欢学 Python。
好好学习，天天向上。我喜欢学 Python。
```

2. Python 程序的注释

(1) 在 Python 程序中，以符号“#”开头的行被当作是注释行。

(2) 在 Python 源码的文件开头中要通过注释行声明编码方式。在中文程序中，一般第 1 行或第 2 行声明编码为“utf-8”。Python 3 新版本已经将源文件的默认编码定为“utf-8”编码，因此程序中也可以省略。

```
#-*- coding: utf-8 -*-
```

(3) 只包含空格和换行符的行，称为空行。Python 3 解释器会自动忽略。

(4) 语句后以符号“#”开始的部分为注释。Python 3 解释器会自动忽略注释部分。

(5) 多行注释的两边用一对 3 个单引号或 3 个双引号包含起来多行文字信息。Python 3 解释器会自动忽略其中的内容。

下面看示例 2-2。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
'''
示例 2-2 注释的演示.py
这是多行注释的演示
'''
#这是单行注释的演示
a=15 #把 15 赋值给变量 a
"""
这是多行注释的演示
a=5 #把 5 赋值给变量 a
"""
print('a=',a)
if a>=10 : #如果 a 大于等于 10
    print('a 大于等于 10! ')
else : #否则，a 小于 10
    print("a 小于 10! ")
```

程序运行结果：

```
a= 15
a 大于等于 10!
```


由于“a=5”这条语句在多行注释中，因此没有被执行。

3. Python 3程序的缩进

Python 3 程序的空格缩进很重要，因为空格缩进长度决定着 Python 3 程序的逻辑结构。

(1) Python 3 用相同空格缩进来表示同一个语句块，缩进是为了方便读取程序，而其他计算机语言一般用花括号“{语句块}”来表示。

(2) Python 3 对缩进空格数没有要求，但在同一个模块中缩进空格数要相同，如都空两个空格作为缩进。

(3) 在 Spyder 编辑器中，“Tab”键表示 4 个空格，因此建议读者以 4 个空格的倍数作语句块划分。

(4) 在 Spyder 编辑器中，按“Tab”键对选中代码块整体增加 4 个空格；按“Shift+Tab”组合键对选中代码块整体减 4 个空格；小于 4 个空格的就不处理了。

在同一个逻辑块中，如果缩进空格数不同，就会引发错误，出现“IndentationError: unexpected indent”。

请看示例 2-3。

```
a=15
if a>=10 :           #如果 a 大于等于 10
    print('a 大于等于 10! ')
else :               #否则，a 小于 10
    print("a 小于 10! ")

b=15
if b<=10 :           #如果 b 大于等于 10
    print('b 小于等于 10! ')
elif b<=20:          #否则，b 小于 10
    print("b 大于 10, 且 b 小于 20! ")
    c=15
    if c==15:
        print('c==5')
elif b<=30:
    print("b 大于 20, 且 b 小于 30! ")
```

程序运行结果:

```
File "C:/示例 2-3.py", line 13
    c=15
    ^
IndentationError: unexpected indent
```

4. Python3程序变量命名规则

(1) 变量名的长度不受限制,但其中的字符必须是字母、数字或者下画线(_),而不能使用空格、连字符、标点符号、引号或其他字符。

(2) 变量名的第一个字符不能是数字,而且必须是字母或下画线。

(3) Python3 区分大小写英文字母。

(4) 不能将 Python3 关键字用作变量名。

通过 dir()函数可以查看目前系统已经定义的名称,包括用户自定义名称。

dir(__builtins__)函数会列出系统内置模块、系统关键字等名称,这些系统内置模块或关键字不能作为变量名或函数名字。

请看示例 2-4。

```
_MyName='Tom'
_My_Name2='Tom2'
a=10
A=20
print('_MyName= ',_MyName)
print('_My_Name2= ',_My_Name2)
print('A= ',A)
print('a= ',a)

print(dir(__builtins__))
```

程序运行结果:

```
_MyName= Tom
_My_Name2= Tom2
A= 20
a= 10
['ArithmeticError', 'AssertionError', 'AttributeError',
'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError',
```

```
'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError',
'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError',
'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError',
'Exception', 'False', 'FileExistsError', 'FileNotFoundError',
'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError',
'ImportError', 'ImportWarning', 'IndentationError', 'IndexError',
'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt',
'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None',
'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError',
'OverflowError', 'PendingDeprecationWarning', 'PermissionError',
'ProcessLookupError', 'RecursionError', 'ReferenceError',
'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',
'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__IPYTHON__',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__',
'__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin',
'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod',
'compile', 'complex', 'copyright', 'credits', 'debugfile', 'delattr', 'dict',
'dir', 'display', 'divmod', 'enumerate', 'eval', 'exec', 'filter', 'float',
'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr',
'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass',
'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min',
'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range',
'repr', 'reversed', 'round', 'runfile', 'set', 'setattr', 'slice', 'sorted',
'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

上面程序中列出的字符串，都是 Python 3 中的关键字。

2.3 Python的赋值语句

Python 3 的赋值语句比较灵活，有多种语句格式。下面看看这些赋值语句的使用方法。

1. 简单赋值

命令格式：

```
变量名=表达式
```

变量名的数据类型根据表达式的运算结果而定。

请看示例 2-5。

```
a=10                #变量 a 的值是整数 10
b=10.5+a*2.3        #变量 b 的值是表达式的计算结果 33.5
c=[1,2,4,5]         #变量 c 是一个列表
d=(1,2,3,4,5)       #变量 d 是元组
e=True              #变量 e 是逻辑变量，值为 True
f=1.5+2.5j          #变量 f 是复数
g='hello Python !'  #变量 g 是字符串
print('a= ',a)
print('b= ',b)
print('c= ',c)
print('d= ',d)
print('e= ',e)
print('f= ',f)
print('g= ',g)
```

程序运行结果略去，读者可自行尝试运行上面代码。

2. 多重赋值

多重赋值命令格式如下：

```
变量名 1=变量名 2=...=变量名 n=表达式
```

对于基本类型来说，如 `int`，分配独立内存空间赋值给相同值，其中一个变量值变化不会影响其他变量值。

对于对象类型来说，如 `Pandas` 的 `DataFrame` 类型，所有变量指向一个对象类型空间地址，其中任何一个变量对 `DataFrame` 类型进行数据处理，其他变量的数据内容也会变化。这种复制方式叫作浅复制。

例如：

```
df1=df2 = ts.get_k_data('000001',ktype='D')
```

df1 和 df2 指向同一个数据，其中任何一个变量数据发生变化，另外一个变量数据也会随之变化，因为它们是同一份数据。

如果想要新建一份数据，就需要用 copy() 函数，这种复制方式称为深复制。

```
df3=df1.copy()
```

此时，df1 的数据变化不会影响 df3 的内容，请看示例 2-6。

```
import HP_global as g
import HP_data as hp
from HP_sys import *

a=b=10.4                                #变量 a 和 b 的值赋值为 10.4
a=a*3                                    #变量 a 的值乘 3

print('a= ',a)                          #显示变量 a 的值
print('b= ',b)                          #显示变量 b 的值, b 值没有随 a 值变化

ds='2017-01-01'                         #开始日期
de='2017-12-30'                         #结束日期
stockn='600521'                        #股票代码, 600521 华海药业
df=hp.get_k_data(stockn,ktype='D',start=ds,end=de,index=False,autype=
'qfq').head(2)                          #获取股票数

df2=df3=df                              #给变量 df2 和 df3 浅复制 df
df4=df.copy()                           #给变量 df4 深复制 df
print('-----下面是 df2 的内容-----')
print(df2)
print('-----下面是 df4 的内容-----')
print(df4)

df3.open=df3.open*4                     #给 df3['open'] 的值放大 4 倍
print('--下面是浅复制 df2 的内容, open 值放大了 4 倍-----')
print(df2)
print('--可以看到浅复制, df3 修改, df2 的值也变化了。')
```

```
print('--下面是深复制 df4 的内容, 深复制 open-----')
print(df4)
print('-可以看到深复制, df4 的值没有变化。')
print('-不难想象, 修改浅复制的 df2 或 df3, 也破坏了原始数据 df。')
print('-----下面是原始数据 df 的内容-----')
print(df)
print('--df 与 df4 比较 open, 原始数据 df 也被修改掉了。')
```

程序运行结果如下:

```
a= 31.2000000000000003
b= 10.4
-----下面是 df2 的内容-----
      date    open  close  high  low  volume  code
0  2017-01-03  18.234  18.234  18.283  18.127  25855.0  600521
1  2017-01-04  18.258  18.193  18.258  18.028  55337.0  600521
-----下面是 df4 的内容-----
      date    open  close  high  low  volume  code
0  2017-01-03  18.234  18.234  18.283  18.127  25855.0  600521
1  2017-01-04  18.258  18.193  18.258  18.028  55337.0  600521
--下面是浅复制 df2 的内容, open 值放大了 4 倍-----
      date    open  close  high  low  volume  code
0  2017-01-03  72.936  18.234  18.283  18.127  25855.0  600521
1  2017-01-04  73.032  18.193  18.258  18.028  55337.0  600521
--可以看到浅复制, df3 修改, df2 的值也变化了。
--下面是深复制 df4 的内容, 深复制 open-----
      date    open  close  high  low  volume  code
0  2017-01-03  18.234  18.234  18.283  18.127  25855.0  600521
1  2017-01-04  18.258  18.193  18.258  18.028  55337.0  600521
-可以看到深复制, df4 的值没有变化。
-不难想象, 修改浅复制的 df2 或 df3, 也破坏了原始数据 df。
-----下面是原始数据 df 的内容-----
      date    open  close  high  low  volume  code
0  2017-01-03  72.936  18.234  18.283  18.127  25855.0  600521
1  2017-01-04  73.032  18.193  18.258  18.028  55337.0  600521
-df 与 df4 比较 open, 原始数据 df 也被修改掉了。
```

上例演示结果表明, 复杂数据类型一定要用深复制, 这样才能避免原始数据被误修改。

3. 多元赋值

多元赋值是指一条赋值语句实现多个变量的赋值过程。例如，“a,b,c=1,2,3”。多元赋值要求变量名的个数和表达式的个数相等，否则会出现错误。

例如，4 个变量仅提供 3 个表达式 “a,b,c,d=1,2,3*5”，就会出现错误信息 “ValueError: not enough values to unpack (expected 4, got 3)”。

又如，3 个变量提供 4 个表达式 “a,b,c=1,2,3*5,18*2”，也会出现错误信息 “ValueError: too many values to unpack (expected 3)”。

多元赋值使程序编写更加简洁、方便，常常被用于函数中，使一个函数返回多个变量值。

请看示例 2-7。

```
a,b,c=1,2,3*5
print('a= ',a)
print('b= ',b)
print('c= ',c)

def xyz(n):
    x=n*n
    y=n^2
    z=n%2
    return x,y,z

x,y,z=xyz(6)
print('x= ',x)
print('y= ',y)
print('z= ',z)
```

程序运行结果：

```
a= 1
b= 2
c= 15

x= 36
y= 4
z= 0
```

4. 增强赋值

增强赋值是指赋值运算符。

例如，“a*=3”相当于“a=a*3”。

请看示例 2-8。

```
a=5
a+=10
print('a=5, a+=10 , a= ',a)

a=35
a-=10
print('a=35, a-=10 , a= ',a)

a=35
a*=10
print('a=35, a*=10 , a= ',a)

a=35
a/=10
print('a=35, a/=10 , a= ',a)

a=35
a%=10
print('a=35, a%=10 , a= ',a)
```

程序运行结果：

```
a=5, a+=10 , a= 15
a=35, a-=10 , a= 25
a=35, a*=10 , a= 350
a=35, a/=10 , a= 3.5
a=35, a%=10 , a= 5
```

2.4 Python的输入语句和输出语句

前面我们学习了 Python 3 的赋值语句，本节介绍其输入语句和输出语句。

1. input 输入函数

Python 3 的输入函数是 `input()` 函数。`input()` 函数从标准输入中读取一个字符串，但对于用户输入的回车键不会读取，因为它是以回车键作为输入结束标志的。

(1) `input()` 函数读取的是一个字符串，如果想要输入数字型，就需要做数据类型的转换。

```
a=input('请输入一个股票代码: ')
print(type(a))
#我们可以使用 int() 函数把字符串转为数字
b=int(input('请输入一个数字: '))
print(type(b))
```

(2) 如果想要输入多个数值变量，则可以使用 `eval()` 函数。

```
#如果是多个字符变量，不能用 int() 函数把字符串转换为数字，就可以使用 eval() 函数
a,b=eval(input('请输入数字 a,b :'))
print(type(a),type(b))
```

下面给出 `input()` 函数的代码，示例 2-9。

```
#input() 函数读取的是一个字符串，如果想要输入数字型，就需要做数据类型的转换
a=input('请输入一个数字: ')
print(a,type(a))
#我们可以使用 int() 函数把字符串转换为数字
b=int(input('请输入一个数字: '))
print(b,type(b))

#如果想要输入多个数值变量，则可以使用 eval() 函数
a,b=eval(input('请输入数字 a,b :'))
print(a,type(a),b,type(b))
```

命令执行结果：

```
请输入一个数字: 20
20 <class 'str'>

请输入一个数字: 30
30 <class 'int'>

请输入数字 a,b :12,34
12 <class 'int'> 34 <class 'int'>
```

2. print 输出函数

Python 2 和 Python 3 的 print 语句用法不同：Python 2 输出用 print 语句，Python 3 输出用 print() 函数。这里只介绍 Python 3 的 print() 函数。

Python 3 的 print() 函数用法如下：

1) 可以输出任何类型的变量或对象

无论什么类型的数据，如字符串、数值、布尔型、列表、字典等都可以直接输出。

示例 2-10。

```
a='Python3 !'
print('Hello '+a)
d = {'a':1, 'b':2}
print('d= ',d)
c=12.5
print('c= ',c)
d=True
print('d= ',d)
```

程序运行结果：

```
Hello Python3 !
d=  {'a': 1, 'b': 2}
c=  12.5
d=  True
```

如果你希望将输出的值转换成字符串，则可以利用 repr() 函数或 str() 函数来实现。

str() 函数：可以将数值数据转换为字符串。

repr() 函数：产生一个易于阅读的字符串格式，参数可以是 Python 3 的任何对象。

下面是示例 2-11。

```
import numpy as np

print(str(10.5),str(2+3j),repr([x for x in range(10)]))
print()
a = np.arange(15).reshape(3, 5)
print(a)
```

```
print()
print(repr(a))
```

命令执行结果:

```
10.5 (2+3j) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

2) 自动换行

Print 函数会自动在行末加上回车, 如果不需要, 则只需要在 print 函数的结尾添加一个“end”参数, 即可改变它的输出行为。例如:

```
for x in range(0,5):
    print (x,end='')
```

在 Python 3 中 print(x)默认是换行的, 如果想插入一个空行也可以写成 print(), 如果不想换行就要写成 print(x, end = ")。

见示例 2-12:

```
for x in range(0,5):
    print (x,end=' ')

print()
for x in range(0,5):
    print (x,end=',')

print()
for x in range(0,5):
    print (x,end='')
```

命令执行结果:

```
0 1 2 3 4
```

```
0,1,2,3,4,  
01234
```

2.5 Python程序流程控制语句

在 Python 程序运行过程中有一个内部命令指针，它指向当前要运行的语句行。因此，Python 程序流程控制语句可以改变程序命令指针的位置。

1. 顺序执行

顺序执行指程序由上至下，逐行逐条执行每行命令。Python 程序默认顺序执行。

2. 条件分支语句

条件分支语句一般指 if 语句。

(1) 单条件分支命令格式。

```
if 逻辑表达式:  
    命令块  
else :  
    命令块
```

(2) 多条件语句格式。

```
if 逻辑表达式 1:  
    命令块  
elif 逻辑表达式 2:  
    命令块  
...  
else :  
    命令块
```

示例 2-13：根据学生分数打印字母等级，程序代码如下。

```
#示例 2-13 根据学生分数打印字母等级.py  
score =float(input('请输入成绩 : '))  
level = int(score / 10)  
  
if level >= 10:
```

```

    print('Level A+')
elif level == 9:
    print('Level A')
elif level == 8:
    print('Level B')
elif level == 7:
    print('Level C')
elif level == 6:
    print('Level D')
else:
    print('Level E')

```

程序运行结果如下：

```

请输入成绩 : 68
Level D

```

(3) 三元条件语句格式。

变量 v=表达式结果 x if 逻辑表达式 A else 表达式结果 y

解释：如果“逻辑表达式 A”为 True，则“表达式结果 x”赋值给变量 v，否则把“表达式结果 y”赋值给变量 v。

下面是示例 2-14。

```

#a 是判定条件
a=True
b=5
c=20

#方法一：为真时的结果 if 判定条件 else 为假时的结果
d = b if a else c      #如果 a 为真，结果是 b，否则结果是 c
print('方法一输出结果：', d)

#方法二：判定条件 and 为真时的结果 or 为假时的结果
d = a and b or c      #如果 a 为真，结果是 b，否则结果是 c
print('方法二输出结果：', d)

#以上两种方法等同于 if ... else ...
if a:

```

```
d = b                #d=b=5
else:
    d = c             #d=c=20
print('if 语句的输出结果: ', d)
```

程序运行结果:

```
方法一输出结果: 5
方法二输出结果: 5
if 语句的输出结果: 5
```

3. 循环语句

Python 有三种循环类型: for 循环、while 循环和内嵌循环。循环体中的控制语句有 break 语句 (循环中断)、continue 语句 (跳过循环体后面的语句开始执行下一轮循环语句块)。

1) for 循环

首先学习 range()函数的用法。Python 3 的 range()函数返回的是一个可迭代对象 (类型是对象) 而不是列表类型, 可用在 for 循环中。函数语法:

```
range(start, stop[, step])
```

start: 计数从 “start” 开始, 默认从 0 开始。例如, range(5)等价于 range(0,5)。

stop: 计数到 “stop” 结束, 但不包括 “stop”。例如, range(0,5)产生的是[0, 1, 2, 3, 4]的序列对象, 没有 5。

step: 步长, 默认为 1。例如, range(0,5)等价于 range(0, 5,1)。

例如, range(10)会产生一个[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]的序列对象。

for 循环类型用于遍历一个序列对象中的所有元素, 如序列数据 (list, tuple, range, str)、集合和映射对象 (如字典) 等。这与其他计算机语言的 for 语句有本质区别。

Python 3 的 for 语句格式如下:

```
for 变量a in 序列对象list :
    循环体语句块
```

下面看一个例子, 打印出福利彩票 3D 的全部号码。我们知道福利彩票 3D 的号码范围为 000~999, 即共有 1000 种不同的号码, 这是一个 0~9 的排列问题, 如示例 2-15 所示。

```

#打印全排列 m
#排列 m
m=3  #=3 3D; =5 排列 5
print()
print('共有排列数为: ',10**m)
# **代表平方
for a in range(10**m):
    print(str(a).zfill(3)) #转换为字符串, 前面补 0

```

程序运行结果:

```

共有排列数为:  1000
000
001
002
003
004
...中间省略
998
999

```

示例 2-16: 通过 for 循环打印九九乘法表。

```

for j in range(1, 10):
    for i in range(1, j+1):
        print('%d*%d=%d' % (i, j, i*j), end='\t')
        i += 1
    print()
    j += 1

```

程序运行结果:

```

1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12  4*4=16
1*5=5  2*5=10  3*5=15  4*5=20  5*5=25
1*6=6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7  2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8  2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9  2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81

```

2) while 循环

Python 的 while 命令格式如下：

```
while 逻辑表达式（也称循环条件）：  
    循环体命令块
```

示例 2-17：通过 while 循环打印九九乘法表。

```
j = 1  
while j <= 9:  
    i = 1  
    while i <= j:  
        print('%d*%d=%d' % (i, j, i*j), end='\t')  
        i += 1  
    print()  
    j += 1
```

程序运行结果：

```
1*1=1  
1*2=2   2*2=4  
1*3=3   2*3=6   3*3=9  
1*4=4   2*4=8   3*4=12   4*4=16  
1*5=5   2*5=10   3*5=15   4*5=20   5*5=25  
1*6=6   2*6=12   3*6=18   4*6=24   5*6=30   6*6=36  
1*7=7   2*7=14   3*7=21   4*7=28   5*7=35   6*7=42   7*7=49  
1*8=8   2*8=16   3*8=24   4*8=32   5*8=40   6*8=48   7*8=56   8*8=64  
1*9=9   2*9=18   3*9=27   4*9=36   5*9=45   6*9=54   7*9=63   8*9=72   9*9=81
```

3) 嵌套循环

各种循环可以嵌套，但是语句块空格需要统一，而完整功能的循环体就是一个语句块。

我们来看一个号码组合的例子，打印出体育彩票 11 选 5 的全部号码组合。从题意中就能看出，这需要 5 层循环，其中每组号码排列有号码重复，组合没有号码重复。

请看示例 2-18。

```
#打印出彩票 11 选 5 的全部号码组合  
#组合 m 选 5
```



```

#m=11,n=5
print()
m=11 #M
hm=[]
for a in range(1, m+3):
    for b in range(a+1, m+1):
        for c in range(b+1, m+1):
            for d in range(c+1, m+1):
                for e in range(d+1, m+1):
                    hm.append([a,b,c,d,e])
                    #print(f,e,d,c,b,a)
print('共有组合数为: ',len(hm))
for x in hm:
    print(x)

```

程序运行结果:

```

共有组合数为:  462
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 6]
[1, 2, 3, 4, 7]
[1, 2, 3, 4, 8]
[1, 2, 3, 4, 9]
[1, 2, 3, 4, 10]
[1, 2, 3, 4, 11]
[1, 2, 3, 5, 6]
...中间略去
[5, 8, 9, 10, 11]
[6, 7, 8, 9, 10]
[6, 7, 8, 9, 11]
[6, 7, 8, 10, 11]
[6, 7, 9, 10, 11]
[6, 8, 9, 10, 11]
[6, 8, 9, 10, 11]
[7, 8, 9, 10, 11]

```

4) 循环体内的分支语句

循环体内也可以有分支语句，但是语句块空格需要统一，而完整功能的分支体就是一个语句块。分支语句和循环语句混合成为一个复杂的语句块。

一般来说，分支语句可以控制循环体内的循环次数。

(1) 满足分支条件的 `break` 语句可以使程序中止当前循环体的运行，使命令运行指针跳出当前循环体，并指向循环体外部的第一条语句行开始执行命令。

请看示例 2-19，打印大于 0 且小于 20 的所有偶数。

```
#打印大于 0 且小于 20 的所有偶数
print()
for a in range(1,20):
    if a%2==0:        #是偶数
        print(a,end=',')
```

程序运行结果：

```
2,4,6,8,10,12,14,16,18,
```

请看示例 2-20。

```
#打印 1-20 的所有偶数，含 20
print()
for a in range(1,100):
    if a%2==0:        #是偶数
        print(a,end=',')
    if a==20:         #如果 a=20，则跳出循环体
        break
```

程序运行结果：

```
2,4,6,8,10,12,14,16,18,20,
```

(2) `continue` 语句可以使程序中止循环体内的 `continue` 语句后面的所有语句块，而接着执行下一轮循环语句块。

请看示例 2-21。

```
#打印 1-20 的所有偶数，含 20
print()
for a in range(1,100):
    if a%2!=0:        #不是偶数
        continue     #继续下一轮循环，后面循环体内的语句都不执行了

    print(a,end=',')
```

```
if a==20: #如果 a=20, 则跳出循环体
    break
```

程序运行结果:

```
2,4,6,8,10,12,14,16,18,20,
```

4. 占位语句

占位语句指 `pass` 语句, 什么也不执行, 只是在语句体中占位。

由于 Python 语法结构的特殊性, 在设计程序时, 针对没有想好的分支语句、循环语句或函数体的命令块, 如果用空格代替, 就会出现语法错误。此时, 我们就可以用 `pass` 语句作为程序结构的占位语句。当想好具体的算法时, 再用算法命令块去代替 `pass` 占位语句。

请看示例 2-22。

```
#打印 1-20 的所有偶数, 含 20
print()
for a in range(1,100):
    if a%2==1: #是奇数
        pass #占个位置
    else:
        print(a,end=', ')
    if a==20: #如果 a=20, 则跳出循环体
        break
```

程序运行结果:

```
2,4,6,8,10,12,14,16,18,20,
```

2.6 import语句

Python 中可以使用 `import` 语句来引入模块、包和库。

`import` 语句的一般语法格式如下:

```
import <模块名|包名|库名>
```

Python 模块就是一个包含了一组定义的变量、函数和类所组成的 Python 文件。

这个文件可以是以.py 为后缀的原始文本文件，也可以是.py 文件编译过的.pyc 文件。

我们编写的程序一般也存储在.py 文件中，如果这个文件包含变量、函数或类，就可以提供给其他 Python 程序使用。假定模块名为 stockn.py。

```
#文件名: stockn.py
#数字股票代码转换字符串股票代码
def stockname(n):
    '''
        数字股票代码转换字符串股票代码
    stockname(n)
        参数: n 整型
        返回: 字符串
    '''
    s=str(n)
    s=s.strip()
    if (len(s)<6 and len(s)>0):
        s=s.zfill(6)+'.SZ'
    if len(s)==6:
        if s[0:1]=='0':
            s=s+'.SZ'
        else:
            s=s+'.SH'
    return s
```

如果我们想用 import 语句导入 stockn.py 模块，就可以使用这个模块中的 stockname(n)函数，模块名可以省略.py，如示例 2-23 所示。

```
import stockn
mystock=650
stock=stockn.stockname(mystock)
stock2=stockn.stockname(600020)
print('stock= ',stock)
print('stock2= ',stock2)
```

程序运行结果：

```
stock= 000650.SZ
stock2= 600020.SH
```

为了便于管理，我们将不同的模块按功能放置到不同的子目录中。如果在子目录“funa”和子目录“funb”中都包含aaa.py模块，那么在导入时就需要增加子目录名。

“funa/aaa.py”文件内容如下：

```
def aaa():
    return 'funa/aaa.py'
```

“funb/aaa.py”文件内容如下：

```
def aaa():
    return 'funb/aaa.py'
```

下面是演示示例 2-24。

```
import funa.aaa
import funb.aaa
aa=funa.aaa.aaa()
print('\naa=funa.aaa.aaa()')
print('aa= ',aa)

aa=funb.aaa.aaa()
print('\naa=funb.aaa.aaa()')
print('aa= ',aa)

from funa.aaa import *
bb=aaa()
print('\nfrom funa.aaa import *')
print('bb= ',bb)

from funb.aaa import *
bb=aaa()
print('\nfrom funb.aaa import *')
print('bb= ',bb)
```

程序运行结果：

```
aa=funa.aaa.aaa()
aa= funa\aaa.py
```

```
aa=funb.aaa.aaa()  
aa= funb\aaa.py  
  
from funa.aaa import *  
bb= funa\aaa.py  
  
from funb.aaa import *  
bb= funb\aaa.py
```

包是一个有层次的文件目录结构，它定义了由 N 个模块或 N 个子包组成的 Python 应用程序执行环境。

包所在子目录与模块子目录功能用法基本一样。包所在目录中一般包含特定的 `__init__.py` 文件，这个文件和其他诸多 `.py` 文件构成一个包。

`__init__.py` 文件可以为空文件，也可以写入一些包执行的初始化代码。

假如 `func` 包中的 `__init__.py` 文件有包执行的初始化代码，那么我们就可以引用这个包名。

```
import func
```

假如 `func` 的目录有 `__init__.py` 和 `aaa.py` 两个文件，其中 `__init__.py` 文件为空文件，那么此时包和带子目录的模块用法就没什么区别。我们用如下格式导入包中的 `aaa.py` 文件。

```
import func.aaa
```

看示例 2-25。

```
from func.aaa import *  
bb=aaa()  
print('\nfromfunc.aaa import *')  
print('bb= ',bb)
```

程序运行结果：

```
from func.aaa import *  
bb= func\aaa.py
```

库是由众多包和模块组成的，是能够完成某类程序开发所必须具有的基础功能。Python 库分为自带内置库（也称标准库）和第三方库。

例如, `math` 是内置数学库, `Tkinter` 是用于开发 Python 图形用户界面的内置库。

第三方库一般都用 `pip` 软件安装管理。例如, `NumPy` 库是一个具有强大科学计算功能的第三方库, `Pandas` 库是建立在 `NumPy` 库上的常用数据处理库。

用户自编模块也可以认为是最小的 Python 自定义库。

使用 `import` 语句引入库有以下几种方法。

(1) 直接引用库。

```
import <库名>
```

此时, 程序可以调用库中的所有函数, 其格式如下:

```
<库名>.<函数名>(<函数参数>)
```

例如, 使用模块 `bbb.py`。

```
#模块 bbb.py
var1='var1'
var2='var2'
var3=18.55

def aaa():
    return 'aaa'

def bbb():
    return 'bbb2'

def ccc():
    return 'ccc3'

def ddd(x='ddd4'):
    return x
```

请看演示示例 2-26。

```
import bbb
bb=bbb.aaa()
print('bb= ',bb)

bb=bbb.ddd()
```

```
print('bb= ',bb)

bb=bbb.ccc()
print('bb= ',bb)
```

程序运行结果:

```
bb=  aaa
bb=  ddd4
bb=  ccc3
```

(2) 直接引用库，并增加别名。

```
import <库名> as <别名>
```

此时，程序可以调用库中的所有函数，其格式如下：

```
<别名>.<函数名>(<函数参数>)
```

请看示例 2-27。

```
import bbb as aa
bb=aa.aaa()
print('bb= ',bb)

bb=aa.ddd()
print('bb= ',bb)

bb=aa.ccc()
print('bb= ',bb)
print('var3= ',aa.var3)
```

程序运行结果:

```
bb=  aaa
bb=  ddd4
bb=  ccc3
var3=  18.55
```

(3) 如果只从库中引用函数，就可以使用如下的命令格式。

```
from <库名> import <函数名 1, 函数名 2, ..., 函数名 n>
```


此时，程序可以调用库中的函数，也可以直接省略库名。使用库中函数的格式如下：

```
<函数名>(<函数参数>)
```

请看示例 2-28。

```
from bbb import bbb,ddd
bb=bbb()
print('bb= ',bb)

bb=ddd('xxx')      #输入参数'xxx'
print('bb= ',bb)

#bb=ccc()          #可删除注释再运行一次
```

程序运行结果：

```
bb=  bbb2
bb=  xxx
```

如果使用模块 bbb.py 中的 ccc 函数，就会出现错误提示：

“NameError: name 'ccc' is not defined”。

(4) 如果不写具体函数名，直接用通配符“*”，则表示引入库中所有函数。在使用中可以省略模块库名，其格式如下：

```
from <库名> import *
```

其中，*是通配符，表示库中所有函数。

此时，在调用该库的函数时不需要使用库名，直接使用如下格式：

```
<函数名>(<函数参数>)
```

请看演示示例 2-29。

```
from bbb import *
bb=bbb()
print('bb= ',bb)

bb=ddd('12345')
print('bb= ',bb)
```

```
bb=ccc()  
print('bb= ',bb)
```

程序运行结果:

```
bb= bbb2  
bb= 12345  
bb= ccc3
```

(5) 库中定义的类和变量的使用方法与库中函数的使用方法相同。

例如:

```
<库名>.<类名>  
<别名>.<变量名>
```

见示例 2-30。

```
import bbb as aa  
bb=aa.ddd('ABCD')  
print('bb= ',bb)  
  
bb=aa.ccc()  
print('bb= ',bb)  
  
print('var3= ',aa.var3)  
  
v1=aa.var1  
v2=aa.var2  
print('v1= ',v1)  
print('v2= ',v2)
```

程序运行结果:

```
bb= ABCD  
bb= ccc3  
var3= 18.55  
v1= var1  
v2= var2
```

(6) 在 Python 中多个模块或程序使用同一个文件, 如果不设定库名或别名, 就会认为是一次使用的局部变量。程序终止, 变量就会释放消失。

3

第 3 章

Python 的数据与运算

一般来说，程序=数据+算法。因此，要想学 Python 编程，首先要了解 Python 的基本数据类型，并掌握它们的用法和操作。

其次，要根据不同数据选择合适的数据类型。例如，在对用户属性的记录中，用户名可以选择字符串类型、年龄可以选整型（整数）、身高可以选浮点数（带小数点的数字）。

最后，要学会不同类型数据的转换，如数字 123 如何转换为字符串'123'等。

3.1 Python的数据类型

Python 3 有 6 个标准的数据类型：

- 数字（Number）。
- 字符串（String）。
- 序列或列表（List）。
- 元组（Tuple）。
- 集合（Set）。
- 字典（Dictionary）。



1. 数字 (Number)

Python 3 支持整数 (int)、浮点数 (float)、布尔值 (bool)、复数 (complex) 4 种数字类型。

1) 整数

整数通常被称为整型，可分为正整数或负整数，不带小数点。Python 3 整型是不限制大小的，可以当作长整型 “Long” 类型使用，所以 Python 3 没有 Python 2 的长整数类型。

整数可以用数字表示，也可以用十六进制或八进制表示。数字表示方法和数学上的写法一样，而不同的电脑系统、编程语言对于十六进制数值有不同的表示方式。Python, Java 及其他相近的语言使用字首 “0x”，如 “0x5A3”，其中开头的 “0” 可使解析器更容易辨认数，而 “x” 则代表十六进制，另外 “o” 代表八进制。在 “0x” 中，“x” 可以大写也可以小写。0x 作为前缀和 0~9, a~f 或 A~F 等来表示十六进制数，如 “0x3F8E00”。下面给出整数示例。

见示例 3-1。

```
a=-12345678
b=0x12345678
c=0o20
print('a=',a)
print('b=',b)
print('b= 0x%x'%b)
print('c=',c)
print('c= 0o%o'%c)
```

命令执行结果：

```
a= -12345678
b= 305419896
b= 0x12345678
c=16
c= 0o20
```

2) 浮点数

浮点数也就是带小数点的数值，浮点数由整数部分与小数部分组成。浮点数可以用数学写法，如 1.23, 3.14, -9.01 等形式来表示，但是对于很大或很小的浮点数，

就必须用科学计数法表示。例如，2.5e3 中的“e”表示数字 10，“e3”表示 10 的 3 次方，因此 $2.5e3 = 2500$ 。下面给出浮点数示例 3-2。

```
a=-12.345678
b=-1.85324534234e8
print('a=',a)
print('b=',b)
print('b= %e'%b)
```

命令执行结果：

```
a= -12.345678
b= -185324534.234
b= -1.853245e+08
```

浮点数可以用 round() 函数来改变小数点后面的位数。例如，`c=round(12.34567894)` 的结果为 `c=12.3456`。

浮点数可以用 int() 函数来取整数。例如，`d=int(12.3456789)` 的结果为 `d=12`。

见示例 3-3。

```
c=round(12.3456789,4)
print('c= ',c)
d=int(12.3456789)
print('d= ',d)
```

命令执行结果：

```
c= 12.3457
d= 12
```

3) 布尔值

Python 3 的布尔值有 True（真）和 False（假），它们与整数中的 1 和 0 有对应关系。下面给出布尔值示例 3-4。

```
a=True
b=False
print('a=',a)
print('b=',b)
print('a==1 ',a==1)
print('b==0 ',b==0)
```

```
print('a==18 ',a==18)
print('a==0 ',a==0)
print('a== -1 ',a== -1)
```

命令执行结果:

```
a= True
b= False
a==1  True
b==0  True
a==18 False
a==0  False
a== -1 False
```

4) 复数

Python 3 的复数类型存放的是一对浮点数, 一个表示实数部分 (real), 另一个表示虚数部分 (imag, 跟随一个字母 j)。

复数由实数部分和虚数部分构成, 可以用 “a + bj” 或者 “complex(a,b)” 表示, 其中实部 “a” 和虚部 “b” 都是浮点型。

例如, 整数 “20”, 浮点数 “3.45+e12”, 复数 “4.53-7.2j”。

除了 “//” “%” 运算符和个别不支持复数的函数外, 其他操作符与函数都可以用于复数操作。

下面给出复数示例 3-5。

```
a=2.912+3.289j
print('复数 a= ',a)
print('实部: ',a.real,' , 虚部: ',a.imag)
print('共轭复数: ',a.conjugate()) #输出该复数的共轭复数
print()
b=complex(1.23,4.56)
print('复数 b= ',b)
print('实部: ',b.real,' , 虚部: ',b.imag)
print('共轭复数: ',b.conjugate()) #输出该复数的共轭复数
```

命令执行结果:

```
复数 a= (2.912+3.289j)
实部: 2.912 , 虚部: 3.289
```

```
共轭复数: (2.912-3.289j)
复数 b= (1.23+4.56j)
实部: 1.23 , 虚部: 4.56
共轭复数: (1.23-4.56j)
```

2. 字符串 (String)

字符串属于序列类型 (sequence type)。字符串是以单引号 “'”、双引号 “””、三个单引号 “'''” 或者三个双引号 “"""” 括起来的任意文本，如'abcd', "1234"等。请注意，单引号、双引号等本身只是一种表示方式，不是字符串的一部分。

如果要表示字符单引号或双引号，就要使用转义符 “\” 来标识。例如，I'm fine ! 用字符串表示为'I\'m fine !'，这个字符串的长度为 10。

字符串的转义字符同 C++的转义字符。见表 3-1。

表 3-1 Python 3 字符串的转义字符

| 转义字符 | 作用描述 |
|------|--------------------|
| \\ | 斜线 “\” |
| \' | 单引号 “'” |
| \" | 双引号 “”” |
| \a | 发出系统响铃声 |
| \b | 退格符 |
| \n | 换行符 |
| \t | 横向制表符 |
| \v | 纵向制表符 |
| \r | 回车符 |
| \f | 换页符 |
| \o | 八进制数代表的字符 |
| \x | 十六进制数代表的字符 |
| \000 | 终止符，\000 后的字符串全部忽略 |

Python 3 使用 len(str)函数来获取字符串 “str” 的长度。

前面介绍了 Python 3 的字符串属于序列类型，其相当于其他计算机语言中的数组类型。因此，Python 3 有序列索引，而其他语言称其为数组下标。Python 3 的序列索引第一个值是 0。例如，在 b='1234'中 b[0]的值是字符 1，b[2]的值是字符 3。

Python 3 的序列类型都可以进行切片操作（slice），即从一个字符串中获取子字符串（字符串的一部分）。我们使用方括号“[]”、起始偏移量“start”、终止偏移量“end”，以及可选的步长“step”来定义一个切片。

格式：序列类型值或序列变量[start:end:step]

下面给出字符串示例 3-6。

```
a='I\'m fine !'
b=len(a)           #获取字符串长度
print('a=',a)
print('b=len(a)=',b)
#字符串转换为浮点数
c='1234'
d=float(c)
print('d=float(c)= ',d)
#字符串切片操作
e='abcdefghijk'
print('e= \'%s\''%e)
print('e[0]= ',e[0])
print('e[-1]= ',e[-1])
print('e[2:4]= ',e[2:4])
print('e[5:]= ',e[5:])
print('e[:5]= ',e[:5])
print('e[::2]= ',e[::2])
f='123456789'
print('f= \'%s\''%f)
print('f[::2]= ',f[::2])
```

命令执行结果：

```
a= I'm fine !
b=len(a)= 10
d=float(c)= 1234.0
e= 'abcdefghijk'
e[0]= a
e[-1]= k
e[2:4]= cd
e[5:]= fg hijk
```



```
e[:5]= abcde
e[::2]= acegik
f= '123456789'
f[::2]= 13579
```

3. 列表 (List)

Python 3 内置的一种数据类型是列表，也称为序列。列表类似于数组，可以随时添加和删除其中的元素。列表是在方括号之间，用逗号分隔开的元素，列表的索引号从 0 开始。

列表中的元素可以是数字类型、字符串类型，也可以是列表数据类型（嵌套列表）。列表元素类型可以不相同，也可以有重复（后面介绍的集合元素无重复）。另外，列表元素有顺序，除非用程序或命令改变顺序（后面介绍的集合元素没有顺序）。

在量化程序中，我们经常用列表来存放股票代码和股票行情数据。例如：

```
stocks=['0000800','300050','600001']
```

4. 元组 (Tuple)

元组是一种有序列表。元组和列表非常类似，但是元组一旦初始化就不能修改。元组使用小括号“()”来包含元素，也用逗号“,”来分隔元素，其索引号也是从 0 开始的。例如：

```
stocks=('000001','600080','300005')
```

5. 集合 (Set)

在集合中，没有重复的元素。另外，由于集合元素没有顺序，因此也不存在集合索引。

在 Python 3 中，集合使用大括号“{}”来包含元素，也用逗号“,”来分隔元素。下面看一个列表、元组、集合的示例，见示例 3-7。

```
stock1=['000001','600080','300005']    #列表
stock2=('000001','600080','300005')    #元组
stock3={'000001','600080','300005'}    #集合
print('stock1= ',stock1,type(stock1))
print('stock1[0]= ',stock1[0])
```

```
print('stock2= ',stock2,type(stock2))
print('stock2[1]= ',stock2[1])
print('stock3= ',stock3,type(stock3))
print(' \'600080\' in stock3 :','600080' in stock3)
```

命令执行结果:

```
stock1= ['000001', '600080', '300005'] <class 'list'>
stock1[0]= 000001
stock2= ('000001', '600080', '300005') <class 'tuple'>
stock2[1]= 600080
stock3= {'300005', '600080', '000001'} <class 'set'>
'600080' in stock3 : True
```

6. 字典 (Dictionary)

由于 Python 3 内置了字典数据类型, 因此其具有极快的查找速度。字典使用大括号 “{}” 来包含元素, 其中元素使用“键值(key): 数值(value)”的形式存储。键值必须是数字、字符串或元组, 这些数据对象不可变动, 如{key1:value1,key2: value2}。

同一个字典的键值不能相同, 如果相同, 后面的数值就会取代前面键值的数据。在设计量化软件中, 我们就可以使用字典数据来存放颜色及对应的 RGB 编码表。下面给出字典的示例 3-8。

```
cla={1:2,2:3,1:5}          #字典
print('原始输入数据 cla={1:2 , 2:3 , 1:5}')
print('实际内容 cla= ',cla)
#颜色编码表
cns = {
'aliceblue':          '#F0F8FF',
'black':               '#000000',
'blue':               '#0000FF',
'brown':               '#A52A2A',
'coral':               '#FF7F50',
'cornflowerblue':     '#6495ED',
'cornsilk':            '#FFF8DC',
'crimson':             '#DC143C',
'cyan':                '#00FFFF',
'darkblue':            '#00008B'}
```

```
print('cns[\'blue\']=      ',cns['blue'])
print('cns[\'darkblue\']=   ',cns['darkblue'])
print('cns[\'cyan\']=       ',cns['cyan'])
```

命令执行结果:

```
原始输入数据 cla={1:2 , 2:3 , 1:5}
实际内容 cla= {1: 5, 2: 3}
cns['blue']=      #0000FF
cns['darkblue']=   #00008B
cns['cyan']=       #00FFFF
```

`len(dict)`中的 `dict` 表示字典数据, 用来计算字典元素个数, 即键的总数。

`str(dict)`可把字典转换成可打印的字符串。

`type(variable)`中的 `variable` 表示变量, 用于返回变量类型。

请看示例 3-9。

```
a={1:2 , 2:3 , 1:5,3:4}
b=a.copy()
print(a==b)
print('len(a)= ',len(a))
print(a)
print('str(b)= ' +str(b))
print('type(b)= ',type(b))
```

程序运行结果:

```
True
len(a)= 3
{1: 5, 2: 3, 3: 4}
str(b)= {1: 5, 2: 3, 3: 4}
type(b)= <class 'dict'>
```

Python 字典包含了以下内置方法:

`dict.clear()`: 删除字典内所有元素。

`dict.copy()`: 返回一个字典的复制。

`dict.fromkeys(seq: val)`: 创建一个新字典, 以序列“seq”中的元素作为字典的键, `val` 为字典所有键对应的初始值。

`dict.get(key, default=None)`: 返回指定键的值。如果值不在字典中, 就返回

default 的值。

`dict.has_key(key)`: 如果键在字典“dict”中就返回 True, 否则返回 False。

`dict.items()`: 以列表返回可遍历的“(键, 值)”元组数组。例如:

```
di={1:2,2:3,3:5}
print(di.items())
```

输出结果:

```
dict_items([(1, 2), (2, 3), (3, 5)])
```

`dict.keys()`: 以列表返回一个字典所有的键。

`radiansdict.setdefault(key, default=None)`: 和`get()`函数类似, 但如果键不存在于字典中, 就会添加键并将值设为 default。

`dict.update(dict2)`: 把字典 dict2 的“(键, 值)”对更新到 dict 中。

`dict.values()`: 以列表返回字典中的所有值。

见示例 3-10。

```
#字典练习
#1.创建空字典
d={}
print('type(d)',type(d))
#2.直接赋值创建
di={1:2,2:3,3:5}
print('原始输入数据 di={1:2 , 2:3 , 3:5}')
print('实际内容 di= ',di)
print(di.items())
#3.通过关键字 dict 和关键字参数创建
dic = dict(spam = 1, egg = 2, bar =3)
print('实际内容 dic= ',dic)
#4.通过二元组列表创建
list = [('spam', 1), ('egg', 2), ('bar', 3)]
dic2 = dict(list)
print('实际内容 dic2= ',dic2)
#5.dict 和 zip 结合创建
dic3 = dict(zip('abc', [1, 2, 3]))
print('实际内容 dic3= ',dic3)
#6.通过字典推导式创建
```

```

dic4 = {i:2*i for i in range(4)}
print('实际内容 dic4= ',dic4)
#7.通过 dict.fromkeys() 函数创建
dic5 = dict.fromkeys(range(4), 'x')
print('实际内容 dic5= ',dic5)
#8.其他方法
list = ['x', 1, 'y', 2, 'z', 3]
dic6 = dict(zip(list[:2], list[1:2]))
print('实际内容 dic6= ',dic6)
#9.radiansdict.clear(): 删除字典内所有元素
di.clear()
print('di.clear()')
print('实际内容 di= ',di)
dica=dic
dicb=dic.copy()
dica.clear()
print('dica.clear()')
print('实际内容 dic= ',dic)
print('实际内容 dicb= ',dicb)

print(' dic4.get(2)= ',dic4.get(2))
print(' dic6.keys()= ',dic6.keys())
print(' dic6.values()= ',dic6.values())

```

程序运行结果:

```

type(d) <class 'dict'>
原始输入数据 di={1:2 , 2:3 , 3:5}
实际内容 di= {1: 2, 2: 3, 3: 5}
dict_items([(1, 2), (2, 3), (3, 5)])
实际内容 dic= {'spam': 1, 'egg': 2, 'bar': 3}
实际内容 dic2= {'spam': 1, 'egg': 2, 'bar': 3}
实际内容 dic3= {'a': 1, 'b': 2, 'c': 3}
实际内容 dic4= {0: 0, 1: 2, 2: 4, 3: 6}
实际内容 dic5= {0: 'x', 1: 'x', 2: 'x', 3: 'x'}
实际内容 dic6= {'x': 1, 'y': 2, 'z': 3}
di.clear()
实际内容 di= {}

```

```
dica.clear()
实际内容 dic= {}
实际内容 dicb= {'spam': 1, 'egg': 2, 'bar': 3}
dic4.get(2)= 4
dic6.keys()= dict_keys(['x', 'y', 'z'])
dic6.values()= dict_values([1, 2, 3])
```

3.2 运算符及优先级

运算符是各类表达式运算的操作符，如算术表达式需要使用算术运算符。

运算符具有优先级，如在算术运算中的“先乘除，后加减”。

Python 的各类运算符也有优先级，在表达式中还可以通过增加小括号“()”来改变优先级的顺序。

1. 算术运算符

算术运算符的种类及意义如表 3-2 所示。

表 3-2 算术运算符

| 运算符 | 表达式 | 描述 | 举例 |
|-----|------|--------------------------|-------------------------------|
| + | x+y | 加——两个对象相加 | 1+3=4; 'abc'+123='abc123' |
| - | x-y | 减——得到负数或是一个数减去另一个数 | 5-3=2 |
| * | x*y | 乘——两个数相乘或是返回一个被重复若干次的字符串 | 2*4=8; 'abc'*3='abcabcabc' |
| / | x/y | 除——x 除以 y | 4/2=2.0 |
| % | x%y | 取模——返回除法的余数 | 20%8=4 |
| ** | x**y | 幂——返回 x 的 y 次幂 | 2**5=32 |
| // | x//y | 取整除——向下取接近除数的整数 | 7//2=3 |

算术运算符也能对一些类型数据操作，如字符串、复数及复合数据类型。例如，Numpy 中的 array 类矩阵数据、Pandas 中的 DataFrame 数据类型。算术运算符代码示例 3-11。

```
#算术运算符示例
import numpy as np
```

```

print('345+678= ',345+678)
print('\ '345\'+\ '678\'= ', '345'+ '678')
print('100+58= ',100+58)
print('2*4= ',2*4)
print('\ 'abc\'*3= ', 'abc'*3)
print('7/2= ',7/2)
print('8/2= ',8/2)
print('20%8= ',20%8)
print('2**5= ',2**5)
print('7//2= ',7//2)
data = [[1,2], [3,4], [5,6]]
print('data=',data)
datax = np.array(data)
print('\ndatax=np.array(data)=\n',datax)
print('\ndatax*2=\n',datax*2)

```

命令执行结果:

```

345+678= 1023
'345'+ '678'= 345678
100+58= 158
2*4= 8
'abc'*3= abcabcab
7/2= 3.5
8/2= 4.0
20%8= 4
2**5= 32
7//2= 3
data= [[1, 2], [3, 4], [5, 6]]
datax=np.array(data)=
[[1 2]
 [3 4]
 [5 6]]
datax*2=
[[ 2  4]
 [ 6  8]
[10 12]]

```

2. 比较运算符

比较运算符的种类及意义如表 3-3 所示。

表 3-3 比较运算符

| 运算符 | 表达式 | 描述 |
|-----|------|---|
| == | x==y | 等于——比较对象是否相等。如果相等则返回 True，否则返回 False |
| != | x!=y | 不等于——比较两个对象是否不相等。如果不相等则返回 True，否则返回 False |
| > | x>y | 大于——返回 x 是否大于 y。如果大于则返回 True，否则返回 False |
| < | x<y | 小于——返回 x 是否小于 y。如果小于则返回 True，否则返回 False |
| >= | x>=y | 大于等于——返回 x 是否大于等于 y。如果大于等于则返回 True，否则返回 False |
| <= | x<=y | 小于等于——返回 x 是否小于等于 y。如果小于等于则返回 True，否则返回 False |

比较运算符代码示例 3-12。

```
#比较运算符示例
import numpy as np
print('28==28 运算结果:',28==28)
print('18==28 运算结果:',18==28)
print('28!=28 运算结果:',28!=28)
print('18!=28 运算结果:',18!=28)
print('28>28 运算结果:',28>28)
print('18>28 运算结果:',18>28)
print('28<28 运算结果:',28<28)
print('18<28 运算结果:',18<28)
print('28>=28 运算结果:',28>=28)
print('18<=28 运算结果:',18<=28)
print('\ 'abc\ '<\ 'bbb\ ' 运算结果:', 'abc'<'bbb')
print('\ 'abc\ '==\ 'bbb\ ' 运算结果:', 'abc'=='bbb')
print('\ 'ccc\ '==\ 'ccc\ ' 运算结果:', 'abc'=='bbb')
print('\ '\ '==None 运算结果:', '\ '==None)
#使用 arange 生成连续元素
a =np.zeros((2,3))
b =np.zeros((2,3))
c = np.ones((2,3))
print('a=\n',a)
print('b=\n',b)
```



```
print('c=\n',c)
print('a==b 运算结果:\n',a==b)
print('a==c 运算结果:\n',a==c)
```

命令执行结果:

```
28==28 运算结果: True
18==28 运算结果: False
28!=28 运算结果: False
18!=28 运算结果: True
28>28 运算结果: False
18>28 运算结果: False
28<28 运算结果: False
18<28 运算结果: True
28>=28 运算结果: True
18<=28 运算结果: True
'abc'<'bbb' 运算结果: True
'abc'=='bbb' 运算结果: False
'ccc'=='ccc' 运算结果: False
''==None 运算结果: False

a=
[[0. 0. 0.]
 [0. 0. 0.]]
b=
[[0. 0. 0.]
 [0. 0. 0.]]
c=
[[1. 1. 1.]
 [1. 1. 1.]]
a==b 运算结果:
[[ True  True  True]
 [ True  True  True]]
a==c 运算结果:
[[False False False]
 [False False False]]
```

3. 赋值运算符

赋值运算符的种类及意义见表 3-4。

表 3-4 赋值运算符

| 运算符 | 示例 | 描述 |
|-----|-----------|--|
| = | c = a + b | 简单的赋值运算符，c = a + b 即将 a + b 的运算结果赋值给 c |
| += | c += a | 加法赋值运算符，c += a 等效于 c = c + a |
| -= | c -= a | 减法赋值运算符，c -= a 等效于 c = c - a |
| *= | c *= a | 乘法赋值运算符，c *= a 等效于 c = c * a |
| /= | c /= a | 除法赋值运算符，c /= a 等效于 c = c / a |
| %= | c %= a | 取模赋值运算符，c %= a 等效于 c = c % a |
| **= | c **= a | 幂赋值运算符，c **= a 等效于 c = c ** a |
| //= | c //= a | 取整除赋值运算符，c //= a 等效于 c = c // a |

赋值运算符代码示例 3-13。

```

a=3
print('a=',a)
b=35
print('b=',b)
c = a + b
print('c=a + b 运算结果: c= ',c)
c += a
print('c += a 运算结果: c= ',c)
c -= b
print('c -= b 运算结果: c= ',c)
c *= a
print('c *= a 运算结果: c= ',c)
c /= a
print('c /= a 运算结果: c= ',c)
b %= a
print('b %= a 运算结果: b= ',b)
b **= a
print('b **= a 运算结果: b= ',b)
b //= a
print('b //= a 运算结果: b= ',b)
    
```

命令执行结果:

```

a= 3
b= 35
    
```

```
c=a + b  运算结果: c= 38
c += a   运算结果: c= 41
c -= b   运算结果: c= 6
c *= a   运算结果: c= 18
c /= a   运算结果: c= 6.0
b %= a   运算结果: b= 2
b **= a  运算结果: b= 8
b //= a  运算结果: b= 2
```

4. 逻辑运算符

逻辑运算又称布尔运算，逻辑运算符的种类及意义如表 3-5 所示。

表 3-5 逻辑运算符

| 运算符 | 逻辑表达式 | 描述 |
|-----|---------|---|
| and | x and y | 布尔“与”。如果 x 为 False 则返回 False，否则返回 y 的计算值 |
| or | x or y | 布尔“或”。如果 x 为 True 则返回 x 的值，否则返回 y 的计算值 |
| not | not x | 布尔“非”。如果 x 为 True 则返回 False；如果 x 为 False 则返回 True |

逻辑运算符代码示例 3-14。

```
a=True
print('a=',a)
b=False
print('b=',b)
c=a and b
print('c=a and b 运算结果: c= ',c)
c=a or b
print('c=a or b 运算结果: c= ',c)
c= not a
print('c=not a 运算结果: c= ',c)
c= not b
print('c=not b 运算结果: c= ',c)
c=30>=20
print('c= 30 >= 20 运算结果: c= ',c)
c= a and 15
print('c= a and 15 运算结果: c= ',c)
c= 15 and a
```

```
print('c= 15 and a 运算结果: c= ',c)
c= b and 15
print('c= b and 15 运算结果: c= ',c)
c= 15 and b
print('c= 15 and b 运算结果: c= ',c)
c= a or 15
print('c= a or 15 运算结果: c= ',c)
c= 15 or a
print('c= 15 or a 运算结果: c= ',c)
c= b or 15
print('c= b or 15 运算结果: c= ',c)
c= 15 or b
print('c= 15 or b 运算结果: c= ',c)
c= a and 'abc'
print('c=a and \'abc\' 运算结果: c= ',c)
```

命令执行结果:

```
a= True
b= False
c=a and b 运算结果: c= False
c=a or b 运算结果: c= True
c=not a 运算结果: c= False
c=not b 运算结果: c= True
c= 30 >= 20 运算结果: c= True
c= a and 15 运算结果: c= 15
c= 15 and a 运算结果: c= True
c= b and 15 运算结果: c= False
c= 15 and b 运算结果: c= False
c= a or 15 运算结果: c= True
c= 15 or a 运算结果: c= 15
c= b or 15 运算结果: c= 15
c= 15 or b 运算结果: c= 15
c=a and 'abc' 运算结果: c= abc
```

5. 位运算符

位运算符的种类及意义如表 3-6 所示。

表 3-6 位运算符

| 运算符 | 表达式 | 描述 | 示例 (a = 60, b = 13) |
|-----|-------|--|--|
| & | a & b | 二进制与：参与运算的两个值，如果两个相应位都为 1，则该位的结果为 1，否则为 0 | (a & b) (二进制表示为 0000 1100) |
| | a b | 二进制或：只要对应的两个二进位有一个为 1，结果位就为 1 | (a b) = 61 (二进制表示为 0011 1101) |
| ^ | a ^ b | 二进制异或：当两个对应的二进位相异时，结果为 1 | (a ^ b) = 49 (二进制表示为 0011 0001) |
| ~ | ~a | 二进制的补：对数据的每个二进制位取反，即把 1 变为 0，把 0 变为 1。~x 类似于 -x-1 | (~a) = -61 (二进制表示为 1100 0011 以 2 的补码形式，由一个带符号二进制数) |
| << | a <<n | 二进制左移：运算数的各二进位全部左移 n 位，“<<”右边的数指定移动的位数，高位丢弃，低位补 0，每移 1 位相当于整数乘 2 | a << 2 = 240 (二进制表示为 1111 0000) |
| >> | a >>n | 二进制右移：把 “>>” 左边的运算数的各二进位全部右移 n 位，“>>” 右边的数指定移动的位数，每移 1 位相当于整数除 2 | a >> 2 = 15 (二进制表示为 0000 1111) |

位运算主要指二进制数据的位运算。在 Python 中，二进制数前加上 “0b”，如 “0b1010”（不是字符串）是十进制的整数 10。十进制数转换成二进制数用 bin(x) 函数，返回的是 “x” 的二进制字符串。

位运算符代码示例 3-15。

```
a = 0b00111100      #60 = 0011 1100
b = 13               #13 = 0000 1101
print ('a=',a,': 二进制',bin(a),'b=',b,': 二进制',bin(b))
c = a & b            #12 = 0000 1100
print ('c = a & b 运算结果: c= ',bin(c))

c = a | b            #61 = 0011 1101
print ('c = a | b 运算结果: c= ',bin(c))
c = a ^ b            #49 = 0011 0001
print ('c = a ^ b 运算结果: c= ',bin(c))
c = ~a               #-61 = 1100 0011
print ('c = ~a 运算结果: c= ',bin(c))
c = a << 2            #240 = 1111 0000
print ('c = a << 2 运算结果: c= ',bin(c))
c = a >> 2            #15 = 0000 1111
print ('c = a >> 2 运算结果: c= ',bin(c))
```

命令执行结果：

```
a= 60 : 二进制 0b111100 b= 13 : 二进制 0b1101
c = a & b 运算结果: c= 0b1100
c = a | b 运算结果: c= 0b111101
c = a ^ b 运算结果: c= 0b110001
c = ~a 运算结果: c= -0b111101
c = a << 2 运算结果: c= 0b11110000
c = a >> 2 运算结果: c= 0b1111
```

6. 成员运算符

除以上运算符之外，Python 还支持成员运算符，其种类及意义如表 3-7 所示。能够进行成员运算的数据类型包括字符串、列表和元组。

表 3-7 成员运算符

| 运算符 | 表达式 | 描述 |
|--------|------------|--|
| in | x in y | 如果在指定的序列中找到值则返回 True，否则返回 False。假如 x 在 y 序列中，“x in y”就返回 True |
| not in | x not in y | 如果在指定的序列中没有找到值则返回 True，否则返回 False。假如 x 不在 y 序列中，“x not in y”就返回 True |

成员运算符代码示例 3-16。

```
a = 10
l = [1, 2, 3, 4, 5] #列表
s = (10, 20, 30, 40, 50) #集合
print('a= ',a)
print('l= ',l)
print('s= ',s)
c= a in l
print('c= a in l 运算结果: c= ',c)
c= a in s
print('c= a in s 运算结果: c= ',c)
c= a not in l
print('c= a not in l 运算结果: c= ',c)
c= a not in s
print('c= a not in s 运算结果: c= ',c)
```

```
b=[] #空列表
print('b=[] #空列表')
c= b not in l
print('c= b in l 运算结果: c= ',c)
c= b in s
print('c= b in s 运算结果: c= ',c)
```

命令执行结果:

```
a= 10
l= [1, 2, 3, 4, 5]
s= (10, 20, 30, 40, 50)
c= a in l 运算结果: c= False
c= a in s 运算结果: c= True
c= a not in l 运算结果: c= True
c= a not in s 运算结果: c= False
b=[] #空列表
c= b in l 运算结果: c= True
c= b in s 运算结果: c= False
```

7. 身份运算符

身份运算符用于比较两个对象的存储单元地址是否相同。除了简单数值、字符串、逻辑变量之外，其他复合数据都是对象数据。对象数据比较的是变量所对应的对象地址，如果它们的对象地址相同，那么这些变量就是使用的同一份数据。如果对其中一个变量的数据进行操作，其他变量的数据也会发生改变。因此，当对这类对象变量进行操作时，为了防止更改原始数据，不能用简单赋值语句进行变量赋值，必须使用数据“copy”的方法。身份运算符的种类及意义如表 3-8 所示。

表 3-8 身份运算符

| 运算符 | 表达式 | 描述 |
|--------|------------|--|
| is | x is y | is 是判断两个标识符是不是引用自同一个对象，如 x is y 类似于 id(x) == id(y)。如果引用的是同一个对象则返回 True，否则返回 False |
| is not | x is not y | is not 是判断两个标识符是不是引用自不同的对象，如 x is not y 类似于 id(x) != id(y)。如果引用的不是同一个对象则返回 True，否则返回 False |

身份运算符示例 3-17。

```
a = 20
b = 20
print ('a=',a, ' : id=',id(a), '\nb=',b, ' : id=',id(b))
c = a is b
print('c= a is b 运算结果: c= ',c)
b=30
print ('a=',a, ' : id=',id(a), '\nb=',b, ' : id=',id(b))
c = a is b
print('c= a is b 运算结果: c= ',c)
x=y=[1,2,3,4,5]
print ('x=',x, ' : id=',id(x), '\ny=',y, ' : id=',id(y))
c = x is y
print('c= x is y 运算结果: c= ',c)
x.append(99) #修改了 x 数据, 后面我们发现 y 的数据也改变了
print('x.append(99) 运算结果: x= ',x)
print('x.append(99) 运算结果: y= ',y)
print()
z=x.copy()
print('z=x.copy() 运算结果: z= ',z)
print ('x=',x, ' : id=',id(x), '\nz=',z, ' : id=',id(z))
c = x is z
print('c= x is z 运算结果: c= ',c)
print()
x.pop(5)
print('x.pop(5) 运算结果: x= ',x)
print('x.pop(5) 运算结果: y= ',y)
print ('x=',x, ' : id=',id(x), '\nz=',z, ' : id=',id(z))
c = y is not z
print('c= y is not z 运算结果: c= ',c)
```

命令执行结果:

```
a= 20 : id= 8791414134400
b= 20 : id= 8791414134400
c= a is b 运算结果: c= True
a= 20 : id= 8791414134400
```



```
b= 30 : id= 8791414134720
c= a is b 运算结果: c= False
x= [1, 2, 3, 4, 5] : id= 326381000
y= [1, 2, 3, 4, 5] : id= 326381000
c= x is y 运算结果: c= True
x.append(99) 运算结果: x= [1, 2, 3, 4, 5, 99]
x.append(99) 运算结果: y= [1, 2, 3, 4, 5, 99]
z=x.copy() 运算结果: z= [1, 2, 3, 4, 5, 99]
x= [1, 2, 3, 4, 5, 99] : id= 326381000
z= [1, 2, 3, 4, 5, 99] : id= 326369864
c= x is z 运算结果: c= False
x.pop(5) 运算结果: x= [1, 2, 3, 4, 5]
x.pop(5) 运算结果: y= [1, 2, 3, 4, 5]
x= [1, 2, 3, 4, 5] : id= 326381000
z= [1, 2, 3, 4, 5, 99] : id= 326369864
c= y is not z 运算结果: c= True
```

8. 运算符优先级

各种类型的运算符都具有优先级。例如，在 if 语句的条件判断表达式中，包含各种不同类型的运算表达式，如算术运算、成员运算、关系运算等，复杂的表达式甚至包含位运算、成员切片等，对这些不同类型的运算符（操作符）都具有优先级规范。优先级高的运算符先进行操作，优先级低的运算符后进行操作。

表 3-9 列出了从低到高优先级的所有运算符。

表 3-9 运算符优先级

| 优先级 | 运算符 | 描述 |
|-----|----------------------|------------|
| 1 | lambda | lambda 表达式 |
| 2 | or | 逻辑（布尔）“或” |
| 3 | and | 逻辑（布尔）“与” |
| 4 | not x | 逻辑（布尔）“非” |
| 5 | in, not in | 成员测试 |
| 6 | is, is not | 同一性（身份）测试 |
| 7 | <, <=, >, >=, !=, == | 比较 |
| 8 | | 按位或 |
| 9 | ^ | 按位异或 |

续表

| 优先级 | 运算符 | 描述 |
|-----|------------------|-----------|
| 10 | & | 按位与 |
| 11 | <<, >> | 移位 |
| 12 | +, - | 加法与减法 |
| 13 | *, /, % | 乘法、除法与取余 |
| 14 | +x, -x | 正负号 |
| 15 | ~x | 按位翻转 |
| 16 | ** | 指数（次幂）运算 |
| 17 | x.attribute | 属性或方法 |
| 18 | x[index] | 下标 |
| 19 | x[index1:index2] | 寻址段或切片 |
| 20 | f(arguments...) | 函数调用 |
| 21 | () | 小括号能提高优先级 |

优先顺序规则如下：

（1）先计算小括号“()”中的表达式，最中间的小括号中的表达式最先计算。

例如，“4*(2+3)”的计算结果为 20。

（2）函数调用。例如，“10*abs(2-8)”的计算结果为 60。

（3）列表等下标和切片计算。例如，“5*‘abcdefg’[3]”和“‘abcdefg’[2:5]”的结果是字符串“ddddd”和“cde”。

（4）属性或方法的“.”运算。例如，“(‘ xy’+‘abc ’).strip()”的结果是字符串“xyabc”。

（5）指数（次幂）运算。

（6）一元算术运算，如~x，+x，-x。

（7）二元算术运算，先乘除，后加减。

（8）二进制运算，如移位运算、位与、位异或、位或。

（9）比较运算，如<，<=，>，>=，!=，==。

（10）身份测试、成员测试运算。

（11）逻辑（布尔）运算。

（12）lambda 表达式。

上述规则中的概念在之后的学习中会用到，读者可先了解。

运算符优先级代码实例 3-18。

#运算符优先级示例 3-18

```
a = 20
b = 5
c = 6
d = 3
print ("a=%d b=%d c=%d d=%d" % (a,b,c,d ))
e = a+b*c/d
print ('e = a+b*c/d = 20+8   运算结果: e= ',e)
e = (a + b) * c / d
print ('e = (a+b)*c/d = 24*6/3 运算结果: e= ',e)
f=a+b*c**d
print ('f=a+b*c**d=20+5%216 运算结果: f= ',f)
g=a+b*c <<2
print ('g=a+b*c <<2 =20+30 <<2 运算结果: g= ',g)
print(4*(2+3))
print(10*abs(2-8))
print(5*'abcdefg'[3])
print('abcdefg'[2:5])
print((' xy'+ 'abc ').strip())
```

命令执行结果:

```
a=20 b=5 c=6 d=3
e = a+b*c/d = 20+8   运算结果: e=   30.0
e = (a+b)*c/d = 24*6/3 运算结果: e=   50.0
f=a+b*c**d=20+5%216 运算结果: f=    25
g=a+b*c <<2 =20+30 <<2 运算结果: g=   200
20
60
dddd
cde
xyabc
```

3.3 数值运算

1. 常量和变量

常量是不会变化的量。Python 3 并没有命名常量,也就是说不能像 C 语言那样给

常量起名字，只能以具体数值来代表常量。例如，整数常量 12，字符串常量'abc'等。

Python 3 的常量包括数字、字符串、布尔值（True，False）、空值（None）。

变量是可以通过变量名访问的计算机内存中的一块区域，它可以存储规定范围内的值，而且值可以改变。基于变量的数据类型，Python 3 解释器会分配指定内存，并决定不同的数据在内存中的存储形式。

变量的命名前面已经介绍过。

del 语句可以删除不再使用的变量，并释放变量所占的内存资源。

见示例 3-19。

```
var1=10                #创建变量 var1
var2='hello'           #创建变量 var2
print('var1=',var1)
print('var2=',var2)
del var1,var2           #删除掉 var1 和 var2 变量，以后再引用就会出错
print('var1=',var1)
```

程序运行结果：

```
var1= 10
var2= hello
    print('var1=',var1)
NameError: name 'var1' is not defined
```

2. 变量赋值

Python 中的每个变量必须赋值才会被创建，赋值不需要先做类型声明，变量的类型就是赋值的数据类型或表达式运算结果的数据类型。

等号“=”用来给变量赋值，其左边是一个变量名，右边是存储在变量中的值。

见示例 3-20。

```
n=20                    #n 是整型变量，值为 20
miles = 1000.0          #miles 是浮点型变量，值为 1000.0
name = "John"           #name 是字符串变量，值为 John
print(n,miles,name)
n=35.5                  #n 的值由 20 变为 35.5，变为浮点类型
miles = miles*2          #miles 是浮点型变量，值变为 2000.0
name = "Tom"            #name 是字符串变量，值变为 Tom
print(n,miles,name)
```

程序运行结果:

```
20 1000.0 John
35.5 2000.0 Tom
```

3. 数值类型转换

当 Python 表达式中包含不同数据类型时,如同时包含浮点数和整数,为了提高精度或者取整数,就需要使用类型转换函数。

`int(x)`是将 `x` 转换为纯整数。

`float(x)`是将 `x` 转换为浮点数。

`round(x[,n])`是改变浮点数小数点后的位数,带四舍五入进位,如 `round(10.456,2)` 返回 10.46。

`complex(x)`是将 `x` 转换成实数部分为 `x` 和虚数部分为零的复数。

`complex(x, y)`是将 `x` 和 `y` 转换成一个实数部分是 `x` 和虚数部分是 `y` 的复数。

见示例 3-21。

```
x=12.88
a=int(x)
b=float(a)
c=complex(x)
print('x=',x,type(x))
print('a=',a,type(a))
print('b=',b,type(b))
print('c=',c,type(c))
```

程序运行结果:

```
x= 12.88 <class 'float'>
a= 12 <class 'int'>
b= 12.0 <class 'float'>
c= (12.88+0j) <class 'complex'>
```

4. 数学函数

Python 的标准数学函数分为内置函数和 `math` 数学库。

(1) 表 3-10 中的是内置函数,其不需要引入库。

表 3-10 内置函数

| 函数 | 功能说明 |
|---------------|--|
| abs(x) | 返回数字的绝对值，如 abs(-10) 返回 10 |
| float(x) | 整数 x 浮点化，如 float(10) 返回浮点数 10.0 |
| pow(x,y[,z]) | 返回 x 的 y 次幂，相当于 x**y，如 pow(2,3) 返回 8；或 x**y%z，如 pow(2,3,3) 返回 2 |
| int(x) | 浮点数 x 整数化，如 int(12.34) 返回整数 12 |
| round(x [,n]) | 返回浮点数 x 的四舍五入值，n 代表舍入到小数点后的位数，省略 n 参数返回整数，如 round(10.8) 返回 11，round(12.34324,2) 返回 12.34 |

示例 3-22：对于炒股的小王来说，假如每周能赚 5%，1 万元本金到年底会有多少资金，获利率达到多少？（除去节假日，1 年按 35 周计算。）

通过分析，我们确定可以利用次幂函数进行计算。下面直接给出 Python 程序：

```
wealth=10000*pow((1+0.05),35)
print('小王到年底有',round(wealth,2),'元资金。')
proportion=(wealth-10000)/10000*100
print('获利率为：',round(proportion,2),'%')
```

程序运行结果：

小王到年底有 55160.15 元资金。
获利率为： 451.6 %

示例 3-23：小红有 10 万元，三年内暂不使用。假定银行 1 年定期利息为 10%，买货币基金年化收益也为 10%，其按日结息。假设三年内银行定期利息和货币基金年化收益不会改变，问小红买货币基金的收益比存银行多多少元？

解题思路：银行是定期，3 年是 3 年复利，货币基金是日复利。下面给出程序代码。

```
Y1=100000*(1+0.1)**3           #Y1 银行 3 年定期本金加收益
Y2=100000*(1+0.1/365)**(365*3) #Y2 货币基金按日结息本金加收益 Y3=Y2-Y1
Y3=Y2-Y1                       #差额
print('小红买货币基金比存银行的收益多',round(Y3,2))
```

程序运行结果：

小红买货币基金比存银行的收益多 1880.33

(2) Python 有 math 数学库，在引入数学库后就可以使用更多的数学函数了。

引入数学模块的语句：

```
import math
```

查看 math 库帮助：

```
help('math')
```

数学库的常量见表 3-11。

表 3-11 math 数学库的常量

| 常量 | 常量说明 |
|---------|----------------------|
| math.e | 返回 2.718281828459045 |
| math.pi | 返回 3.141592653589793 |

数学库的函数见表 3-12。

表 3-12 数学库的函数

| 函数 | 功能说明 |
|------------------------|--|
| math.sqrt(x) | 返回数字的平方根，数字可以为负数，返回类型为浮点数，如 math.sqrt(9)返回 9 |
| math.pow(x,y) | 返回 x 的 y 次幂，如 math.pow(2,3)返回 8.0 |
| math.fabs(x) | 返回数字的绝对值，如 math.fabs(-10)返回 10.0 |
| math.fmod(x, y) | 返回 x%y（取余），如 math.fmod(5,2)返回 1.0 |
| math.fsum([x, y, ...]) | 返回无损精度的和，如 math.fsum([0.1, 0.2, 0.3])返回 6.0 |
| math.factorial(x) | 返回 x 的阶乘，如 math.factorial(5)返回 120 |
| math.copysign(x, y) | 若 y<0，则返回-1 乘以 x 的绝对值；否则，返回 x 的绝对值 |
| math.frexp(x) | 返回 m 和 i 元组，满足 x 等于 m 乘以 2 的 i 次方 |
| math.ldexp(m, i) | 返回 m 乘以 2 的 i 次方 |
| math.ceil(x) | 返回数的上入整数，如 math.ceil(4.1)返回 5 |
| math.floor(x) | 返回数的下舍整数，如 math.floor(4.9)返回 4 |
| math.trunc(x) | 返回 x 的整数部分，如 math.trunc(5.8)返回 5 |
| math.modf(x) | 返回 x 的小数和整数，如 math.modf(5.2)返回(0.20000000000000018, 5.0) |
| math.exp(x) | 返回 e 的 x 次方，如 math.exp(2)返回 7.38905609893065 |
| math.expm1(x) | 返回 e 的 x 次方减 1，如 math.expm1(2)返回 6.38905609893065 |
| math.log(x[,base]) | 返回以 e 或 base 为基数的 x 的对数值，如 math.log(math.e)返回 1.0，math.log(100,10)返回 2.0 |
| math.log10(x) | 返回以 10 为基数的 x 的对数，如 math.log10(100)返回 2.0 |
| math.log1p(x) | 返回 1+x 的自然对数（以 e 为底），如 math.log1p(1)返回 0.6931471805599453 |

续表

| 函数 | 功能说明 |
|-------------------------------|--|
| <code>math.degrees(x)</code> | 将弧度转换为角度 |
| <code>math.radians(x)</code> | 将角度转换为弧度 |
| <code>math.sin(x)</code> | 返回正弦值，如 <code>math.sin(math.pi/2)</code> 返回 1.0 |
| <code>math.asin(x)</code> | 返回 x 的反三角正弦值 |
| <code>math.cos(x)</code> | 返回余弦值，如 <code>math.cos(math.pi)</code> 返回 -1.0 |
| <code>math.acos(x)</code> | 返回 x 的反三角余弦值 |
| <code>math.tan(x)</code> | 返回正切值，如 <code>math.tan(1.0)</code> 返回 1.5574077246549023 |
| <code>math.atan(x)</code> | 返回 x 的反三角正切值 |
| <code>math.atan2(x, y)</code> | 返回 x/y 的反三角正切值 |
| <code>math.sinh(x)</code> | 返回 x 的双曲正弦函数 |
| <code>math.asinh(x)</code> | 返回 x 的反双曲正弦函数 |
| <code>math.cosh(x)</code> | 返回 x 的双曲余弦函数 |
| <code>math.acosh(x)</code> | 返回 x 的反双曲余弦函数 |
| <code>math.tanh(x)</code> | 返回 x 的双曲正切函数 |
| <code>math.atanh(x)</code> | 返回 x 的反双曲正切函数 |
| <code>math.hypot(x, y)</code> | 返回以 x 和 y 为直角边的斜边长，如 <code>math.hypot(3,4)</code> 返回 5 |
| <code>math.erf(x)</code> | 返回 x 的误差函数 |
| <code>math.erfc(x)</code> | 返回 x 的余误差函数 |
| <code>math.gamma(x)</code> | 返回 x 的伽玛函数 |
| <code>math.lgamma(x)</code> | 返回 x 的绝对值的自然对数的伽玛函数 |
| <code>math.isinf(x)</code> | 若 x 为无穷大，则返回 True；否则，返回 False |
| <code>math.isnan(x)</code> | 若 x 不是数字，则返回 True；否则，返回 False |

示例 3-24：有一个三位的数字，其数字从 0~9 中随机选取，可重复取数，问每组号码的概率是多少？

解题思路：位置 A，B，C 可选取的数字都是 0~9，有 10 种取法，因此全部组合 P 为 $10 \times 10 \times 10 = 1000$ 。这实际上就是福利 3D 彩票每注奖号的中奖概率。

```
#三位数字 ABC, 每个位置有 10 种选号方法。
A=10
B=10
C=10
D=A*B*C      #D 是全部组合数
P=1/D         #P 是每组号码的概率
print('三位数字 ABC 每组号码的概率: ', round(P, 3))
```


程序运行结果:

三位数字 ABC 每组号码的概率: 0.001

示例 3-25: 有一个五位的数字, 数字从 0~9 中随机选取, 不可重复取数, 问五位数字的组合有多少种?

解题思路: 从题意中可以看出, 这是一个排列问题, 我们可以直接使用排列公式进行计算。

排列 P, 从 n 个元素中取 r 个进行排列, 排列有顺序。排列 P 的组合数为 $n!/(n-r)!$, 其中感叹号 “!” 表示阶乘。我们可以用 `math.factorial(x)` 函数来计算阶乘。程序代码如下:

```
import math
P5=math.factorial(10)/math.factorial(10-5) #P5 五位数字组合数
print('五位数字的组合有',int(P5),'种。')
```

程序运行结果:

五位数字的组合有 30240 种。

示例 3-26: 福利双色球彩票有红球 33 个, 蓝球 16 个, 每注彩票号码由 6 个红球和 1 个蓝球构成, 号码没有顺序, 问所有双色球组合数是多少?

解题思路: 这是一个组合问题, 我们可以直接代入组合公式。组合 C, 从 n 个元素中取 r 个元素, 不进行排列, 组合数为 $C=n!/(r!*(n-r)!)$, 程序如下:

```
import math
R6=math.factorial(33)/(math.factorial(6)*math.factorial(33-6))
    #红球组合数
B1=16    #蓝球取法 16 选 1
C=R6*B1
print('所有双色球组合数是',int(C),'种。')
```

程序运行结果:

所有双色球组合数是 17721088 种。

5. 随机函数

随机函数包括基本随机数函数和扩展随机数函数, 需要使用 Python 标准库——`random`。

引入随机数库语句：

```
import random
```

查看随机数库帮助信息：

```
help(' random ')
```

random 库中的函数见表 3-13。

表 3-13 random 库中的函数

| 函数 | 功能说明 |
|--------------------|---|
| seed(a=None) | 初始化给定的随机种子，默认为当前系统时间。通过 seed() 函数生成的随机数是假的随机数，也称为稳定随机数，是指返回的随机数是一个稳定算法所得出的稳定结果序列。若种子相同，后续所有的随机数就都一样 |
| random() | 生成一个[0.0,1.0)的随机小数 |
| randint(a,b) | 生成[a,b]的整数 |
| randrange(a,b[,k]) | 生成一个[a,b)以 k 为步长的随机整数 |
| getrandbits(k) | 生成 k 比特长的随机整数 |
| uniform() | 生成[a,b]的随机小数 |
| choice(s) | 从序列 s 中随机选择一个元素 |
| shuffle(s) | 将序列 s 重新排列，并返回打乱后的序列，即给列表洗牌 |

随机函数的用处很多，特别是在电子游戏和仿真领域。如果我们想实现一些功能，利用其他语言需要自己写程序，而利用 Python 语言就可以直接使用现成的函数。

示例 3-27：用 Python 实现机选 5 注福利彩票双色球的号码。要想实现这一功能，其他编程语言需要写很多行代码，而 Python 只需要下面几行代码。

```
"""
模拟双色球开奖程序
"""
import random
print()
rb=[x for x in range(1,34)]      #建立一个 1~33 的数字列表 rb
lb=[x for x in range(1,17)]     #建立一个 1~16 的数字列表 lb
random.seed()                   #初始化随机种子
print('机选五注双色球彩票号码：')
for i in range(5):              #我们用 Python 选 5 注号
```

```

random.shuffle(rb)          #用函数 random.shuffle 打乱列表顺序
random.shuffle(lb)          #用函数 random.shuffle 打乱列表顺序
hh=[rb[0],rb[1],rb[2],rb[3],rb[4],rb[5]]
hh.sort()                   #对选中红号排序
print(hh[0],hh[1],hh[2],hh[3],hh[4],hh[5], ' + ', lb[0])

```

程序运行结果:

机选五注双色球彩票号码:

```

6 9 18 20 25 27 + 14
3 7 16 19 24 31 + 9
1 7 8 25 28 31 + 15
2 3 6 23 28 32 + 2
3 11 14 20 22 26 + 1

```

上面例子使用了列表数据类型。

3.4 字符串及相关操作

字符串是 Python 中最常用的数据类型,属于序列类型。Python 的很多特殊应用都依赖于字符串。

1. 字符串和字符串变量

我们可以使用一对单引号 “'” 或双引号 “"” 来创建字符串。

存储字符串的变量,被称为字符串变量。字符串变量也是对象,对象都有方法(函数),因此,一个变量成为字符串变量就会拥有相关字符串处理的方法。例如,示例 3-28。

```

var1='Hello'
var2='World'
var3=var1+var2          #var3 的值为 HelloWorld
var4=var1[0]            #var4 的值 H
print('var1=',var1)
print('var2=',var2)
print('var3=',var3)
print('var4=',var4)

```

程序运行结果：

```
var1= Hello
var2= World
var3= HelloWorld
var4= H
```

下面我们学习有关字符串的处理。

2. 字符串处理

字符串处理函数或方法的种类及意义见表 3-14。

表 3-14 字符串处理函数或方法

| 函数或方法（string 表示字符串） | 说明 |
|--|---|
| len(string) | 返回字符串长度的函数 |
| string.capitalize() | 把字符串的第一个字符大写 |
| string.center(width) | 返回一个原字符串居中，并使用空格填充至长度 width 的新字符串 |
| string.count(str, beg=0, end=len(string)) | 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数 |
| string.encode(encoding='UTF-8', errors='strict') | 以 encoding 指定的编码格式编码 string，如果出错就默认报一个 ValueError 的异常，除非 errors 指定的是“ignore”或者“replace” |
| string.endswith(obj, beg=0, end=len(string)) | 检查字符串是否以 obj 结束，如果 beg 或者 end 指定则检查指定的范围内是否以 obj 结束。如果是则返回 True, 否则返回 False |
| string.expandtabs(tabsize=8) | 把字符串 string 中的 tab 符号转为空格，tab 符号默认的空格数是 8 |
| s.find(str, beg=0, end=len(string)) | 检测 str 是否包含在 string 中，如果 beg 和 end 指定范围则检查是否包含在指定范围内。如果是则返回开始的索引值，否则返回-1 |
| string.format() | 格式化字符串 |
| string.index(str, beg=0, end=len(string)) | 跟 find()方法一样，只不过如果 str 不在 string 中就会报一个异常 |
| string.isalnum() | 如果 string 至少有一个字符并且所有字符都是字母或数字则返回 True，否则返回 False |
| string.isalpha() | 如果 string 至少有一个字符并且所有字符都是字母则返回 True，否则返回 False |
| string.isdecimal() | 如果 string 只包含十进制数字则返回 True，否则返回 False |
| string.isdigit() | 如果 string 只包含数字则返回 True，否则返回 False |

续表

| 函数或方法 (string 表示字符串) | 说明 |
|---|---|
| <code>string.islower()</code> | 如果 <code>string</code> 中包含至少一个区分大小写的字符并且所有这些字符 (区分大小写) 都是小写则返回 <code>True</code> , 否则返回 <code>False</code> |
| <code>string.isnumeric()</code> | 如果 <code>string</code> 中只包含数字字符则返回 <code>True</code> , 否则返回 <code>False</code> |
| <code>string.isspace()</code> | 如果 <code>string</code> 中只包含空格则返回 <code>True</code> , 否则返回 <code>False</code> |
| <code>string.istitle()</code> | 如果 <code>string</code> 是标题化的 (见 <code>title()</code> 函数) 则返回 <code>True</code> , 否则返回 <code>False</code> |
| <code>string.isupper()</code> | 如果 <code>string</code> 中包含至少一个区分大小写的字符, 并且所有这些字符 (区分大小写) 都是大写则返回 <code>True</code> , 否则返回 <code>False</code> |
| <code>string.join(seq)</code> | 以 <code>string</code> 作为分隔符, 将列表 <code>seq</code> 中所有的字符元素合并为一个新的字符串, 如 <code>a=['1','2','3']</code> , <code>b='abc'</code> , <code>b.join(a)</code> 返回 <code>'1abc2abc3'</code> |
| <code>string.ljust(width)</code> | 返回一个原字符串左对齐, 并使用空格填充至长度 <code>width</code> 的新字符串 |
| <code>string.lower()</code> | 转换 <code>string</code> 中所有大写字符为小写 |
| <code>string.lstrip()</code> | 截掉 <code>string</code> 左边的空格 |
| <code>string.maketrans(intab, outtab)</code> | <code>maketrans()</code> 函数用于创建字符映射的转换表, 对于接受两个参数的最简单的调用方式, 第一个参数是字符串表示需要转换的字符, 第二个参数也是字符串表示转换的目标 |
| <code>max(string)</code> | 返回字符串 <code>string</code> 中最大的字母的函数 |
| <code>min(string)</code> | 返回字符串 <code>string</code> 中最小的字母的函数 |
| <code>string.partition(str)</code> | 有点像 <code>find()</code> 和 <code>split()</code> 的结合体, 从 <code>str</code> 出现的第一个位置起, 把字符串 <code>string</code> 分成一个 3 元素的元组 (<code>string_pre_str</code> , <code>str</code> , <code>string_post_str</code>), 如果 <code>string</code> 中不包含 <code>str</code> 则为 <code>string_pre_str == string</code> |
| <code>string.replace(str1, str2, num=string.count(str1))</code> | 把 <code>string</code> 中的 <code>str1</code> 替换成 <code>str2</code> , 如果 <code>num</code> 指定则替换不超过 <code>num</code> 次 |
| <code>string.rfind(str, beg=0, end=len(string))</code> | 类似于 <code>find()</code> , 不过是从右边开始查找 |
| <code>string.rindex(str, beg=0, end=len(string))</code> | 类似于 <code>index()</code> , 不过是从右边开始查找 |
| <code>string.rjust(width)</code> | 返回一个原字符串右对齐, 并使用空格填充至长度 <code>width</code> 的新字符串 |
| <code>string.rpartition(str)</code> | 类似于 <code>partition()</code> , 不过是从右边开始查找 |
| <code>string.rstrip()</code> | 删除 <code>string</code> 字符串末尾的空格 |
| <code>string.split(str="", num=string.count(str))</code> | 以 <code>str</code> 为分隔符切片 <code>string</code> , 如果 <code>num</code> 有指定值, 则仅分隔 <code>num</code> 个子字符串 |
| <code>string.splitlines([keepends])</code> | 按照行 “ <code>\r</code> ”, “ <code>\r\n</code> ”, “ <code>\n</code> ” 分隔, 返回一个包含各行作为元素的列表, 如果参数 <code>keepends</code> 为 <code>False</code> 则不包含换行符, 如果为 <code>True</code> 则保留换行符 |
| <code>string.startswith(obj, beg=0, end=len(string))</code> | 检查字符串是否以 <code>obj</code> 开头, 若是则返回 <code>True</code> , 否则返回 <code>False</code> 。如果 <code>beg</code> 和 <code>end</code> 指定值, 则在指定范围内检查 |

续表

| 函数或方法（string 表示字符串） | 说明 |
|-------------------------------|--|
| string.strip([obj]) | 在 string 上执行 lstrip()和 rstrip() |
| string.swapcase() | 翻转 string 中的大小写 |
| string.title() | 返回“标题化”的 string，就是说所有单词都是以大写开始，其余字母均为小写（见 istitle()） |
| string.translate(str, del='') | 根据 str 给出的表（包含 256 个字符）转换 string 的字符，要过滤掉的字符放到 del 参数中 |
| string.upper() | 转换 string 中的小写字母为大写 |
| string.zfill(width) | 返回长度为 width 的字符串，原字符串 string 右对齐，前面填充 0 |

示例 3-29：将任何格式的中国股市的股票代码转换为聚宽股票代码格式。

相关知识：中国股市的股票代码是 6 位数字，中国股市分为上海证券市场和深圳证券市场，它们有不同的证券品种，如表 3-15 和表 3-16 所示。表 3-17 所示的是全国中小企业股份转让系统品种。

表 3-15 上海证券市场的交易品种

| 上海证券品种 | 代码 |
|--------|--------------------|
| 上海 A 股 | 格式 6×××××，如 600030 |
| 上海 B 股 | 格式 9×××××，如 900901 |
| 上海基金 | 格式 5×××××，如 519002 |
| 上海债券 | 格式 01××××，如 010107 |

表 3-16 深圳证券市场的交易品种

| 深圳证券品种 | 代码 |
|--------|----------------------------------|
| 深圳 A 股 | 格式 00××××，如 000001，001979 |
| 深圳 B 股 | 格式 20××××，如 200011，200992 |
| 深圳基金 | 格式 15××××，如 150008 |
| 深圳债券 | 格式 10××××，11××××，如 100303，118861 |
| 中小板 | 格式 002×××，如 002001，002940 |
| 创业板 | 格式 3××××，如 300001，300760 |

表 3-17 全国中小企业股份转让系统品种

| 转股证券品种 | 代码 |
|--------|---------------------------|
| 新三板 | 格式 4×××××，如 400008，400070 |

聚宽证券代码标准格式（表 3-18），仅适用于 A 股市场股票代码和基金代码，不含债券。

表 3-18 聚宽证券代码标准格式

| 交易市场 | 代码后缀 | 示例代码 | 证券简称 |
|---------|-------|---------------|------|
| 上海证券交易所 | .XSHG | '600519.XSHG' | 贵州茅台 |
| 深圳证券交易所 | .XSHE | '000001.XSHE' | 平安银行 |

解题思路：

（1）首先，我们只考虑 A 股、B 股、中小板、创业板及基金。其次，要考虑到输入股票代码的各种格式，如'000001', 'SZ000001', '000001SZ', '000001.sz', '000001.XSHE', '88'等。我们利用函数从各种字符串中提取出 0~9 的数字，并合成只有数字构成的新字符串。

（2）使用 zfill()方法把只含数字的字符串变为长度为 6 的新字符串，前面补 0。

（3）分析股票代码所处的市场，然后增加对应市场的后缀。

下面给出程序代码示例 3-29。

```
#ston()函数提取字符串中的数字,并返回
#自定义函数概念和用法后面对应的章节会涉及
def ston(string):
    s1=''
    for s in string:
        if s.isdecimal():
            s1=s1+s
    return s1
#聚宽股票代码转换
def jqsn(s):
    s=s.strip()
    s=ston(s)
    if (len(s)<6 and len(s)>0):
        s=s.zfill(6)+'.XSHE'
    if len(s)==6:
        if s[0:1]=='6' or s[0:1]=='9' or s[0:1]=='5':
            s=s+'.XSHG'
        else:
            s=s+'.XSHE'
```

```
return s
print('sz1 ==>',jqsn('sz1'))
print('1.sz ==>',jqsn('1.sz'))
print('400011 ==>',jqsn('400011'))
print('SH600011 ==>',jqsn('SH600011'))
print('900011.sh ==>',jqsn('900011.sh'))
print('500011.XSHG ==>',jqsn('500011.XSHG'))
```

程序运行结果:

```
sz1 ==> 000001.XSHE
1.sz ==> 000001.XSHE
400011 ==> 400011.XSHE
SH600011 ==> 600011.XSHG
900011.sh ==> 900011.XSHG
500011.XSHG ==> 500011.XSHG
```

3. 字符串运算

下面介绍字符串常见的运算方法。

(1) 使用加号“+”可以将两个字符串连接到一起,成为一个新的字符串。例如:

```
tring='abcd'+'efg'    #结果为'abcdefg'
```

(2) 使用乘号“*”可以将一个字符串的内容复制数次,成为一个新的字符串。
例如:

```
s='#'*3+'Python'+'#'*3    #结果为'###Python###'
```

示例 3-30: 隐藏电话号码中第 4 位到第 7 位的数字,换为星号“*”。

```
tela = '18578755056'
telb = tela[:3]+'*'*4+tela[7:]
print(telb)
```

程序运行结果:

```
185****5056
```

(3) 使用大于“>”、大于等于“>=”、等于“==”、小于“<”、小于等于“<=”等逻辑运算符比较 2 个字符串的大小。例如示例 3-31。


```

a='abc'<='aaa'           #结果 False
name='Tom'               #给 name 变量赋值'Tom'
b=name=='Tom'            #判断 name 是否为'Tom', 结果为 True
print('a=',a)
print('b=',b)

```

程序运行结果:

```

a= False
b= True

```

(4) 使用“in”或“not in”关键字测试某个字符串是否存在于另一个字符串内。
例如示例 3-32:

```

x='ab' in 'abc'           #结果为 True
print('x= ',x)
a='2abcd'                 #给变量 a 赋值'2abcd'
f=lambda x:x[0] in '0123456789' #匿名函数 f, 判断字符串首字母是否为数字
y=f(a)                   #返回结果为 True
print('y= ',y)

```

程序运行结果:

```

x= True
y= True

```

示例 3-33: 输入用户名和密码, 判断用户名是否为 administrator, 密码是否为 admin888。如果正确则显示信息“欢迎管理员登录!”, 如果错误则显示“用户名或密码错误!”。用户名从第一个字母开始能连续输入 5 个字符及以上就算正确。判断完, 程序结束。

解题思路: 这是一个普通用户登录的程序段, 可以使用连等于符号(==)来比较。由于用户名比较长, 要求从第一个字母开始能够连续输入 5 个及以上字符, 所以我们使用 rfind() 方法来处理, 示例 3-33 程序代码如下。

```

name=input('请输入用户名:')
password=input('请输入密码:')
if password=='admin888' and 'administrator'.rfind(name)==0 and len
(name)>=5:
    print('欢迎管理员登录! ')
else:
    print('用户名或密码错误! ')

```

程序运行结果:

```
请输入用户名:admin
请输入密码:admin888
欢迎管理员登录!
请输入用户名:admi
请输入密码:asmin888
用户名或密码错误!
```

4. 字符串格式化

Python 3 的字符串格式化有两种方法: %格式符方法和 format 方法。

1) %格式符方法

字符串格式控制:

```
%[(name)][flags][width].[precision]typecode
```

其中, 参数的含义见表 3-19。

表 3-19 字符串格式控制的参数

| 参数 | 说明 |
|-----------|--|
| name | 可为空, 数字 (占位), 命名 (传递参数名, 不能以数字开头) 以字典格式映射格式化, 其为键名 |
| flag | 标记格式限定符号, 包含 “+” “-” “#” 和 “0”, +表示右对齐 (会显示正负号), -表示左对齐, 前面默认为填充空格 (即默认右对齐), 0 表示填充 0, #表示在八进制时前面补充 0、十六进制填充 0x、二进制填充 0b |
| width | 宽度 (最短长度, 包含小数点, 在小于 width 时会填充) |
| precision | 小数点后的位数, 与 C 语言相同 |
| typecode | 输入格式类型 |

表 3-19 中为 typecode 格式组成, Python 3 字符串的格式类型见表 3-20。

表 3-20 Python 3 字符串的格式类型

| type 符号 | 描述 |
|---------|-----------------|
| % | 百分号 (%) 标记 |
| c | 格式化字符及其 ASCII 码 |
| s | 格式化字符串 |
| d | 有符号十进制整数 |

续表

| type 符号 | 描述 |
|---------|-----------------------|
| u | 无符号十进制整数 |
| o | 格式化无符号八进制数 |
| x | 格式化无符号十六进制数 |
| X | 格式化无符号十六进制数（大写字符） |
| f | 格式化浮点数字，可指定小数点后的精度 |
| e | 用科学计数法格式化浮点数 |
| E | 用科学计数法格式化浮点数，用 E 代替 e |
| g | 浮点数字（根据值的大小采用%e 或%f） |
| G | 浮点数字（根据值的大小采用%E 或%f） |
| p | 指针，用十六进制数格式化变量的地址 |

示例 3-34 是关于 Python 3 字符串的格式类型的示例。

```
a=50
print('变量 a 的 ASCII 码%c: %c'%a)
s='Hello World !'
print("The length of %s is %d" % (s,len(s)))
b=-10.8
print('b=-10.8')
print("变量 b 的值为%d: %d"% b)
print("变量 b 的值为%f: %f"% b)
print("变量 b 的值为%u: %u"% b)
c=10/3000
print('c=10/3000')
print("变量 c 的值为%f: %f"% c)
print("变量 c 的值为%e: %e"% c)
print("变量 c 的值为%g: %g"% c)
d=-2304123
print('d=-2304123')
print("变量 d 的值为%u: %u"% d)
print("变量 d 的值为%o: %o"% d)
print("变量 d 的值为%x: %x"% d)
print("变量 d 的值为%X: %X"% d)
```

程序运行结果:

```
变量 a 的 ASCII 码%c: 2
The length of Hello World ! is 13
b=-10.8
变量 b 的值为%d: -10
变量 b 的值为%f: -10.800000
变量 b 的值为%u: -10
c=10/3000
变量 c 的值为%f: 0.003333
变量 c 的值为%e: 3.333333e-03
变量 c 的值为%g: 0.00333333
d=-2304123
变量 d 的值为%u: -2304123
变量 d 的值为%o: -10624173
变量 d 的值为%x: -23287b
变量 d 的值为%X: -23287B
```

示例 3-35:

```
a='股票代码: %d 股票名称: %s 股票价格: %.2f 元'
b=(600519, '贵州茅台', 551.65)
c=a%b
print(c)
e = '★'*5+ '%(string)+20s★★★★★'%{'string': '书山有路勤为径'}
print(e)
f = '★'*5+ '%(string)-20s★★★★★'%{'string': '书中自有黄金屋'}
print(f)
```

程序运行结果:

```
股票代码: 600519 股票名称: 贵州茅台 股票价格: 551.65 元
★★★★★          书山有路勤为径★★★★★
★★★★★书中自有黄金屋          ★★★★★
```

2) format 方法

语法格式:

```
<{模板字符串}>.format(<参数表>)
```

在调用 `format()`方法后会按模板字符串返回一个新的字符串。

模板字符串格式：`{i:<格式控制模板>}`，其中，`i` 是参数表中第 `i` 个参数，参数从 0 开始编号。

格式控制模板：

`[[fill]align][sign][#][0][width][,][.precision][type]`

格式控制模板主要包括 “`[fill][align][sign][width][.precision][type]`” 六个字段，这些字段都是可选的，也可以组合使用。模板参数说明见表 3-21。

表 3-21 `format` 方式的控制模板参数说明

| 参数 | 说明 |
|------------|---|
| fill | 可选。空白处填充的字符 |
| align | 可选。对齐方式，需配合 width 使用。只能选<, >, =, ^四种符号。 <: 内容左对齐。 >: 内容右对齐（默认）。 =: 内容右对齐。将符号放置在填充字符的左侧，且只对数字类型有效。 ^: 内容居中 |
| sign | 可选。有无符号数字，只能选正号“+”，负号“-”和空格。 +: 正号加正，负号加负。 -: 正号不变，负号加负。 空格: 正号空格，负号加负 |
| # | 可选。对于二进制、八进制、十六进制，如果加上#，则会显示 0b/0o/0x，否则不显示 |
| , | 可选。为数字添加分隔符，如 1,000,000 |
| width | 可选。格式化位所占宽度 |
| .precision | 可选。小数位保留精度 |
| type | 可选。格式化类型，同字符串的格式类型，如 d, .3f 等，不需要输入百分符号 |

见示例 3-36:

```
#默认数据顺序{}
aa='股票代码: {} 股票名称: {} 股票价格: {} 元'.format(600519,'贵州茅台',
551.65)
print(aa)
#指定数据顺序{i}
bb='股票代码: {2} 股票名称: {0} 股票价格: {1} 元'.format('贵州茅台',
551.65,600519)
```

```
print(bb)
cc='PI={0:.2f}'.format(3.1415926)
print(cc)
```

程序运行结果:

```
股票代码: 600519  股票名称: 贵州茅台  股票价格: 551.65 元
股票代码: 600519  股票名称: 贵州茅台  股票价格: 551.65 元
PI=3.14
```

3.5 列表及相关操作

列表也称序列, 是 Python 中使用最频繁的数据类型。列表中的每个元素都分配一个数字, 称列表位置, 列表元素也称列表成员。在列表索引中, 第一个索引是 0, 第二个索引是 1, 依此类推。

1. 创建列表

前面我们给大家介绍了字符串数据就是一个存放单字母的特殊列表, 这里介绍的列表可以存放任何数据, 而且列表允许有相同的元素。

我们可以用列表推导式来创建列表, 其格式如下:

```
变量 var = [<生成元素表达式> for <元素 x> in <元素列表 list> if <过滤条件>]
```

例如 (示例 3-37):

```
multiples = [i for i in range(30) if i % 3 is 0]
print(multiples)
```

程序运行结果:

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
```

创建列表, 示例 3-38。

```
#列表变量一般用方括号[]或list()函数创建
list1 = ['Google', '股票', 600088, 20.88, 20.88]
list2 =[x for x in range(10)] #列表推导式
print(list1)
print(list2)
```

程序运行结果:

```
['Google', '股票', 600088, 20.88, 20.88]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2. 访问列表中的元素

要访问列表中的元素，需要使用方括号连同索引或索引切片获得索引对应可用的值，见示例 3-39。

```
#访问列表中的元素
list1 = ['Google', '股票', 600088, 20.88, 20.88]
list2 = [x for x in range(10)] #列表推导式
print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
```

程序运行结果:

```
list1[0]: Google
list2[1:5]: [1, 2, 3, 4]
```

3. 修改列表中的元素

由于列表有索引号，即下标，所以可以通过索引号来修改列表中的元素，见示例 3-40。

```
#修改列表中的元素
list1 = ['Google', '股票', 600088, 20.88, 20.88]
list1[1]=600080
print("list1 修改元素: ",list1)
```

程序运行结果：列表元素“股票”更新为 600080。

```
list1 修改元素: ['Google', 600080, 600088, 20.88, 20.88]
```

4. 增加列表中的元素

列表 `append()` 方法可以增加列表元素，见示例 3-41。

```
#增加列表元素
list1.append('Hepu')
```

```
list1.append(2775205)
print("list1 追加元素: ", list1)
```

程序运行结果:

```
list1 追加元素:  ['Google', 600080, 600088, 20.88, 20.88, 'Hepu', 2775205]
```

5. 列表中插入元素

列表 `insert()` 方法可以插入新元素。插入后，当前索引号及后面的元素后移，见示例 3-42。

```
#插入列表元素
list1=['Google', 600080, 600088, 20.88, 20.88, 'Hepu', 2775205]
list1.insert(5, 600080)
print ('list1 插入元素: ', list1)
```

程序运行结果:

```
list1 插入元素:  ['Google', 600080, 600088, 20.88, 20.88, 600080, 'Hepu',
2775205]
```

6. 删除列表中的元素

删除列表中的元素有两种方式：按值删除和按索引号删除。

1) 按值删除

使用列表 `remove()` 方法从表头开始删除第 1 个值相同的元素，后面相同的值还存在。例如，`list1.remove(600080)`。

2) 按索引号

使用 `del` 语句按索引号删除列表中的元素。例如，`del list1[0]`，见示例 3-43。

```
list1=['Google', 600080, 600088, 20.88, 20.88, 600080, 'Hepu', 2775205]
print ('删除 600080 前为 : ', list1)
list1.remove(600080)
print ('删除第 1 个 600080 后为 : ', list1)
list1.remove(600080)
print ('删除第 2 个 600080 后为 : ', list1)
del list1[0]
print ('del list1[0]后为 : ', list1)
```


程序运行结果：删除前，在列表 list1 中存在两个相同的元素 600080。

```
删除 600080 前为 : ['Google', 600080, 600088, 20.88, 20.88, 600080, 'Hepu',
2775205]
删除第 1 个 600080 后为 : ['Google', 600088, 20.88, 20.88, 600080, 'Hepu',
2775205]
删除第 2 个 600080 后为 : ['Google', 600088, 20.88, 20.88, 'Hepu', 2775205]
del list1[0]后为 : [600088, 20.88, 20.88, 'Hepu', 2775205]
```

7. 列表相关操作

1) 列表运算符 “+” 和 “*”

在列表中，“+” 和 “*” 运算符的作用与字符串相似，其中 “+” 号用于组合列表，“*” 号用于重复列表。

例如示例 3-44。

```
a=[1,2,3,4]
b=['a','b','c']
c=a+b
print('c=a+b= ',c)
d=a*5
print('d=',d)
```

程序运行结果：

```
c=a+b= [1, 2, 3, 4, 'a', 'b', 'c']
d= [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

2) 列表切片

列表切片的操作也与字符串相似。

切片格式：

```
列表[start:[end:[step]]]
```

其中，中间的两个中括号表示可省略的参数，start 表示开始位置，end 表示结束位置，step 表示取值步长。

下面看示例 3-45。

```
b=[x for x in range(10)]
print('b= ',b)
```

```
print('b[0]=' ,b[0])
print('b[2]=' ,b[2])
print('b[-2]=' ,b[-2])
print('b[:5]= ' ,b[:5])
print('b[5:]=' ,b[5:])
print('b[-3:]=' ,b[-3:])
print('b[:-3]=' ,b[:-3])
print('b[4:7]=' ,b[4:7])
print('b[0:10:3]=' ,b[0:10:3])
```

程序运行结果:

```
b= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
b[0]= 0
b[2]= 2
b[-2]= 8
b[:5]= [0, 1, 2, 3, 4]
b[5:] = [5, 6, 7, 8, 9]
b[-3:] = [7, 8, 9]
b[:-3] = [0, 1, 2, 3, 4, 5, 6]
b[4:7] = [4, 5, 6]
b[0:10:3] = [0, 3, 6, 9]
```

3) 列表 in 语句

列表 in 语句用来判断一个元素是否存在列表中。如果存在则返回 True，如果不存在则返回 False。见示例 3-46。

```
#列表 in 语句
print('2 in [1, 2, 3] 结果 ',2 in [1, 2, 3])
print('5 in [1, 2, 3] 结果 ',5 in [1, 2, 3])
```

程序运行结果:

```
2 in [1, 2, 3] 结果  True
5 in [1, 2, 3] 结果  False
```

4) 列表比较符 “==”

列表比较符 “==” 能够用来比较两个列表是否相同。如果相同则返回 True，否则返回 False。见示例 3-47。

```
#列表比较符==
```

```
a='abc'
b="abc"
f= a==b
g= a!=b
print('列表比较 f= ',f)
print('列表比较 g= ',g)
```

程序结果如下:

```
列表比较 f= True
列表比较 g= False
```

5) 列表嵌套

列表能够嵌套。嵌套列表就是列表中的元素也是一个列表。见示例 3-48。

```
#列表合并，列表操作符+
a=['a',80,12.8]
b=[1,2,3]
c=a+b
print('c= a+b = ',c)
#列表操作符*
d=[1,2]
e=d*3    #注意不是列表元素值*3，是列表元素个数*3
print('e= d*3 = ',e)
#列表 in 语句
print('2 in [1, 2, 3] 结果 ',2 in [1, 2, 3])
#列表切片
print('切片 c[2] ',c[2])
print('切片 c[-2] ',c[-2])
print('切片 c[1:] ',c[1:])
#列表比较符==
f= a==b
print('列表比较 f= a==b = ',f)
#嵌套列表
x = ['a', 'b', 'c']
y = [1, 2, 3]
z = [x, y]
print('嵌套列表 z= ',z)
```

程序运行结果：

```
c= a+b= ['a', 80, 12.8, 1, 2, 3]
e= d*3 = [1, 2, 1, 2, 1, 2]
2 in [1, 2, 3] 结果 True
切片 c[2] 12.8
切片 c[-2] 2
切片 c[1:] [80, 12.8, 1, 2, 3]
列表比较 f= a==b = False
嵌套列表 z= [['a', 'b', 'c'], [1, 2, 3]]
```

8. 列表相关函数和方法

列表相关函数的种类及意义见表 3-22。

表 3-22 列表相关函数和方法

| 函数和方法 | 功能 |
|--|-----------------------------------|
| len(list) | 返回列表元素个数的函数 |
| max(list) | 返回列表元素最大值的函数 |
| min(list) | 返回列表元素最小值的函数 |
| list(seq) | 将元组转换为列表的函数 |
| list.append(obj) | 在列表末尾添加新的对象 |
| list.count(obj) | 统计某个元素在列表中出现的次数 |
| list.extend(seq) | 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表） |
| list.index(obj) | 从列表中找出某个值第一个匹配项的索引位置 |
| list.insert(index, obj) | 将对象插入列表 index 位置 |
| list.pop([index=-1]) | 移除列表中的一个元素（默认最后一个元素），并且返回该元素的值 |
| list.remove(obj) | 移除列表中某个值的第一个匹配项 |
| list.reverse() | 反向列表中的元素 |
| list.clear() | 清空列表 |
| list.copy() | 复制列表 |
| list.sort(cmp=None, key=None, reverse=False) | 对原列表进行排序 |

9. 列表迭代操作

Python 的迭代操作就是利用列表有序特性进行迭代输出。

迭代器是一个可以记住遍历位置的对象。

迭代器从列表的第一个元素开始访问，直到所有的元素被访问完，而且只能往前不会后退。字符串、列表和元组对象都可用于创建迭代器。

迭代器有两个基本函数：`iter()`函数和 `next()`函数。

`iter()`函数：创建迭代器对象，并且指针指向第一个元素。

`next()`函数：输出当前指针元素，并且使迭代器指针指向下一个元素。

一般我们用 `for` 语句作列表遍历操作。可参见示例 3-49。

```
list1=[x for x in range(10)]
#使用 for 循环
for i in list1:
    print(i, end=" ")
```

如果使用迭代器，就可以使用 `while` 语句作列表遍历操作。见示例 3-50。

```
list1=[x for x in range(10)]
#使用迭代器，用 while 循环
it = iter(list1)
i=0
while i<len(list1):
    print (next(it), end=" ")
    i+=1
```

上面两个程序运行的结果是相同的。

上面复杂的循环操作，只需要一行 `for` 循环语句就能实现。见示例 3-51。

```
list1=[x for x in range(10)]
#使用 for 循环
print('使用 for 循环')
for i in list1:
    print(i*2, end=" ")
print()
#使用迭代器，用 while 循环
print('使用迭代器，用 while 循环')
it = iter(list1)
i=0
while i<len(list1):
    print (next(it)*2, end=" ")
```

```
i+=1
print()
#for 循环语句迭代命令
print('for 循环语句命令')
for x in list1: x=x*2;print(x, end=" ")
```

程序运行结果:

```
使用 for 循环
0 2 4 6 8 10 12 14 16 18
使用迭代器, 用 while 循环
0 2 4 6 8 10 12 14 16 18
for 循环语句命令
0 2 4 6 8 10 12 14 16 18
```

3.6 集合及相关操作

1. 创建集合

集合 (set) 是一个无序的不重复元素序列, 集合元素也称为集合成员。集合中的元素没有重复, 且没有顺序, 也没有下标。

我们可以使用大括号 “{}” 或者 set() 函数创建集合。注意: 因为 “{}” 可用来创建一个空字典, 因此创建一个空集合必须用 set() 函数, 不能用 “{}”。

创建集合格式:

```
var= set(x)
```

如果不提供任何参数, 则默认会生成空集合。如果提供一个参数 x, 则该参数 x 必须是可迭代的, 即一个序列, 或迭代器, 或支持迭代的一个对象。例如, 一个列表或一个字典。

见示例 3-52。

```
a=set('alacazam')
print('a= ',a)
```

程序运行结果:

```
a= {'m', 'l', 'a', 'z', 'c'}
```

类似于列表推导式，同样，集合也支持集合推导式。见示例 3-53。

```
b = {x for x in 'abracadabra' if x not in 'abc'}
print('b= ',b)
```

程序运行结果：

```
b=  {'r', 'd'}
```

2. 增加集合元素

增加单个集合元素可以使用 `add(x)` 方法，其中参数 `x` 是一个元素值。

格式：

```
sets.add(x)
```

`sets` 表示一个字典。在上述操作中，如果元素已存在，则不进行任何操作。

添加多个集合元素可以使用 `update(x)` 方法，其中参数 `x` 可以是集合、列表、元组、字典等。

格式：

```
sets.update(x)
```

见示例 3-54。

```
a=set('alacazam')
print('a= ',a)
a.add(10)
a.add(20)
print('a= ',a)
a.update({'x','r','z'})
print('a= ',a)
```

程序运行结果：

```
a=  {'m', 'l', 'a', 'z', 'c'}
a=  {'m', 'l', 10, 'a', 'z', 20, 'c'}
a=  {'r', 'm', 'l', 10, 'a', 'x', 'z', 20, 'c'}
```

3. 删除集合元素

删除集合元素可以使用 `remove(x)` 方法，其中 `x` 是一个元素值。

格式:

```
sets.remove(x)
```

将元素 x 从集合 sets 中删除, 如果元素不存在, 则会发生错误。

此外, 还有一个方法也能移除集合中的元素, 且如果元素不存在, 也不会发生错误。

格式:

```
sets.discard(x)
```

因为集合中的元素没有顺序, 所以也可以用 pop() 函数随机删除集合中的一个元素。一般会删除显示顺序的第一个元素, 并返回删除掉的元素值。

格式:

```
var=sets.pop()
```

见示例 3-55。

```
#删除集合元素
b={1,2,3,5}
b.discard(2)
print('删除 b 中元素 2: ',b)
b.discard(2)
print('删除 b 中元素 2 (没有报错): ',b)
b.remove(3)
print('删除 b 中元素 3: ',b)
#弹出一个集合元素
d=b.pop()
print('弹出一个集合元素:',d,' b= ',b)
```

程序运行结果:

```
删除 b 中元素 2:  {1, 3, 5}
删除 b 中元素 2 (没有报错):  {1, 3, 5}
删除 b 中元素 3:  {1, 5}
弹出一个集合元素: 1  b=  {5}
```



4. 集合复制

集合是对象数据，而对象数据不能通过变量来直接赋值。例如，假设 `b` 是集合，`e=b`，则 `id(e)`和 `id(b)`的地址相同。如果修改集合 `e`，则集合 `b` 的数据也随之发生变化，因为集合 `e` 和集合 `b` 是同一份数据。

集合复制要用 `copy()`方法。

格式：

```
var=sets.copy()
```

此时，集合 `var` 和集合 `sets` 的地址不同，即 `id(var)`和 `id(sets)`不相同。此时如果修改集合 `var`，则集合 `sets` 的数据不会发生变化，因为集合 `var` 和集合 `sets` 是不同的数据。

见示例 3-56。

```
a=set('alacazam')
b=a
c=a.copy()
b.add(999)
print('a= ',a)
print('c= ',c)
```

程序运行结果：

```
a= {999, 'm', 'l', 'a', 'z', 'c'}
c= {'z', 'm', 'l', 'c', 'a'}
```

5. 不可变集合

有时为了防止程序意外修改集合数据，可以使用冰冻函数即使用 `frozenset()`函数返回一个不可变的集合备份。我们只能访问，不能修改不可变的集合中的元素。`set()`函数和 `frozenset()`函数可以分别用来生成可变的集合和不可变的集合。

格式：

```
var= frozenset(x)
```

如果想解冻集合就可以使用集合的 `set()`函数。见示例 3-57。

```
bb = {x for x in range(1,5)}
```

```
var = frozenset(bb)
print(var, type(var))
var2 = set(var)
print(var2, type(var2))
```

程序运行结果:

```
frozenset({1, 2, 3, 4}) <class 'frozenset'>
{1, 2, 3, 4} <class 'set'>
```

6. 集合元素个数

格式:

```
var=len(sets)
```

返回集合 sets 元素的个数。

见示例 3-58。

```
bb = {x for x in range(1,5)}
print('bb= ',bb)
print('len(bb)=',len(bb))
```

程序运行结果:

```
bb= {1, 2, 3, 4}
len(bb)= 4
```

7. 清空集合

格式:

```
sets.clear()
```

相关演示见示例 3-59。

```
"""
集合
"""
bb = {x for x in range(1,5)}
var = frozenset(bb)
print(var, type(var))
```

```

var2 = set(var)
print(var2,type(var2))
print()
#可以使用大括号{ }或者 set() 函数创建集合
a=set('alacazam')
print('a= ',a)
#集合支持集合推导式
b = {x for x in 'abracadabra' if x not in 'abc'}
print('b= ',b)
#增加单个集合元素用 add(x) 函数, 其中参数 x 是一个元素值
b.add( 'abc' )
print('增加 abc :b= ',b)
#添加多个集合元素用 update(x) 函数, 其中参数 x 可以是集合、列表、元组、字典等
c={1,2,3}
b.update(c)
print('增加 c={1,2,3} :b= ',b)
#删除集合元素
b.discard(2)
print('删除 b 中元素 2: ',b)
b.discard(2)
print('删除 b 中元素 2 (没有报错): ',b)
b.remove(3)
print('删除 b 中元素 3: ',b)
#弹出一个集合元素
d=b.pop()
print('弹出一个集合元素:',d,' b= ',b)
#集合复制
e=b #把 b 的地址复制给了 e
e.add('xyz')
print('e=b e.add(\'xyz\') b= ',b)
print('id(b)=',id(b),'id(e)=',id(e),' 地址相同')
f=b.copy()
print('f=d.copy() f= ',f)
print('id(b)=',id(b),'id(f)=',id(f),' 地址不相同')

#清空集合
b.clear()
print('清空集合 b= ',b)

```

程序运行结果：

```
a= {'a', 'l', 'm', 'c', 'z'}
b= {'r', 'd'}
增加 abc :b= {'r', 'd', 'abc'}
增加 c={1,2,3} :b= {1, 'd', 2, 3, 'abc', 'r'}
删除 b 中元素 2: {1, 'd', 3, 'abc', 'r'}
删除 b 中元素 2 (没有报错): {1, 'd', 3, 'abc', 'r'}
删除 b 中元素 3: {1, 'd', 'abc', 'r'}
弹出一个集合元素: 1 b= {'d', 'abc', 'r'}
e=b e.add('xyz') b= {'d', 'abc', 'r', 'xyz'}
id(b)= 42507400 id(e)= 42507400 地址相同
f=d.copy() f= {'r', 'd', 'xyz', 'abc'}
id(b)= 42507400 id(f)= 156994024 地址不相同
清空集合 b= set()
```

8. 集合的相关操作

集合的基本功能包括关系测试和消除重复元素。集合对象还支持联合“union”、交集“intersection”、差集“difference”和对称差集“symmetric difference”等数学运算。

集合支持 `x in set`, `len(set)` 和 `for x in set` 等操作，这些操作同列表一样。作为一个无序的集合，其不记录元素的位置和插入点，因此，集合不支持索引下标等类似于列表的操作。

`obj in s` 成员测试：如果 `obj` 是 `s` 中的一个元素则返回 `Ture`，否则返回 `False`。

`obj not in s` 非成员测试：如果 `obj` 不是 `s` 中的一个元素则返回 `Ture`，否则返回 `False`。

`t == s` 集合等价测试：测试集合 `s` 和集合 `t` 是否有相同的元素，如果有则返回 `Ture`，否则返回 `False`。

`t != s` 集合不等价测试：与等价测试相反。

集合 `t` 和集合 `s` 支持一系列标准操作，包括并集、交集、差集和对称差集。例如：

```
a = t | s    #t 和 s 的并集
b = t & s    #t 和 s 的交集
c = t - s    #求差集（项在 t 中，但不在 s 中）
d = t ^ s    #对称差集（项在 t 或 s 中，但不会同时出现在二者中）
```

9. 集合相关函数和方法

集合相关函数和方法的种类及功能见表 3-23。

表 3-23 集合相关函数和方法

| 函数和方法 | 功能 |
|------------------------------|--|
| s.add(obj) | 集合添加成员操作，将成员 obj 添加到集合 s 中 |
| s.remove(obj) | 集合删除成员操作，将成员 obj 从集合 s 中删除。如果集合 s 中不存在该成员，则引发错误 |
| s.discard(obj) | 集合删除成员操作，将成员 obj 从集合 s 中删除。如果集合 s 中不存在该成员，则不提示错误 |
| s.pop() | 移除集合 s 中的任意一个元素并返回 |
| len(s) | 集合 s 中元素的个数的函数 |
| s.clear() | 清除操作。清除集合 s 中的所有元素，s 成为空集合 |
| set([obj]) | 生成可变集合的函数 |
| frozenset([obj]) | 生成不可变集合的函数 |
| obj in s | 成员测试语句 |
| obj not in s | 非成员测试语句 |
| s.isdisjoint(t) 或 s == t | 等价测试 |
| s != t | 不等价测试 |
| s & t | 严格意义上的子集测试，不允许成员相同。当 s!=t 且集合 s 中所有的元素都是集合 t 中的元素 |
| s.issubset(t) 或 s <= t | 子集测试。集合 s 中所有的元素都是集合 t 的成员，允许成员相同 |
| s > t | 严格意义上的超集测试，不允许成员相同。当 s != t 且集合 t 中所有的元素都是集合 s 的成员 |
| s.issuperset(t) 或 s >= t | 超集测试。集合 t 中所有的元素都是集合 s 的成员 |
| s.union(t) 或 s t | 集合合并操作 |
| s.intersection(t) 或 s ^ t | 交集操作，同时是集合 s 和集合 t 中的元素 |
| s.difference(t) 或 s - t | 集合差分操作，是 s 中的元素，不是 t 中的元素 |
| s.copy() | 集合复制操作，返回集合 s 的副本 |

续表

| 函数和方法 | 功能 |
|---|--|
| <code>s.update(t)</code> 或 <code>s = t</code> | 将集合 <code>t</code> 中的成员添加到集合 <code>s</code> 中 |
| <code>s.intersection_update(t)</code> 或 <code>s &= t</code> | 修改集合操作，使集合 <code>s</code> 中仅包括集合 <code>s</code> 和集合 <code>t</code> 中相同的成员 |
| <code>s.difference_update(t)</code> 或 <code>s -= t</code> | 集合差修改操作，使集合 <code>s</code> 中包括仅属于集合 <code>s</code> 但不属于集合 <code>t</code> 的成员 |
| <code>symmetric_difference_update()</code> 或 <code>s ^= t</code> | 对称差分修改操作，集合 <code>s</code> 中包括仅属于集合 <code>s</code> 或仅属于集合 <code>t</code> 的成员 |

10. 集合和列表转换

用 `set()` 函数可以把列表转换为集合，格式如下。

```
sets=set(lists)
```

用 `list()` 函数可以把集合转换为列表，格式如下。

```
lists=list(sets)
```

利用集合的无重复特性，我们可以将股票代码转化为集合，并能快速完成股票代码去重、股票代码池的合并、剔除股票黑名单等运算。见示例 3-60。

```
#建立集合 st1,st2
st1=set(['002027', '600061', '600080','600659'])
print('st1= ',st1)
st2=set(['000001', '600061', '600088','000002'])
print('st2= ',st2)
#集合中增加元素
print('\n----st1---- st1 中增加 600090')
st1.add('600090')
print(st1)
print('\n----a = st1 | st2-----')
#集合运算
a = st1 | st2          #st1 和 st2 的并集
print(a)
print('\n----b = st1 & st2 -----')
b = st1 & st2          #st1 和 st2 的交集
```

```

print(b)
print('\n----c = st1 - st2 -----')
c = st1 - st2      #求差集（项在 st1 中，但不在 st2 中）
print(c)
print('\n----d = st1 ^ st2-----')
d = st1 ^ st2      #对称差集（项在 st1 或 st2 中，但不会同时出现在二者中）
print(d)
print('\n----st2 remove 600088-----')
#集合元素删除
st2.discard('600088')
print('st2 = ',st2)
print('\n----集合转列表-----')
st3=list(st2)
print('st3 = ',st3)
print('st2: ',type(st2),' st3: ',type(st3))

```

程序运行结果:

```

st1= {'600080', '002027', '600659', '600061'}
st2= {'600088', '000001', '000002', '600061'}
----st1---- st1 中增加 600090
{'600659', '600061', '600080', '002027', '600090'}
----a = st1 | st2-----
{'600659', '600061', '600088', '000001', '600080', '002027', '600090',
'000002'}
----b = st1 & st2 -----
{'600061'}
----c = st1 - st2 -----
{'600080', '002027', '600659', '600090'}

----d = st1 ^ st2-----
{'600659', '600088', '000001', '600080', '002027', '600090', '000002'}
----st2 remove 600088-----
st2 = {'000001', '000002', '600061'}
----集合转列表-----
st3 = ['000001', '000002', '600061']
st2: <class 'set'> st3: <class 'list'>

```

4

第 4 章

自定义函数、类和作用域

4.1 Python的自定义函数

前面我们学习了 Python 系统中的函数，这节课主要介绍用户自定义函数的使用。

1. 函数的定义

函数是组织好的，可重复使用的，用来实现相对单一功能或相关联功能的代码段。

在 Python 3 中，无论是命名函数，还是匿名函数，都是语句和表达式的集合。使用函数的过程称为调用函数。

以下是定义一个函数的简单规则：

- (1) 函数代码块以“def”关键词开头，后接函数标识符名称和圆括号“()”。
- (2) 任何传入参数和自变量都必须放在圆括号内，圆括号之间也可以用于定义参数，称为形式参数。
- (3) 函数内容以冒号“:”开始，并且以空格缩进。
- (4) 函数的第一行语句可以选择性地使用注释字符串文档，这个文档用于存放函数说明。
- (5) 使用“return [表达式]”结束函数，选择性地返回一个值给调用方。不带表



达式的“return”相当于返回 None 值。

下面是一个定义函数“def”语法。

```
def function_name([参数表]):
    '''函数介绍文档。'''
    函数命令块
    return [表达式参数表]
```

函数调用是指在程序中执行函数。在函数调用中输入的参数称为实参，如 range(12) 中的 12。

看示例 4-1 中的 stockname(n) 函数定义。

```
#文件名：示例 4-1.py
#数字股票代码转换字符串股票代码
def stockname(n):
    '''
    函数说明
    数字股票代码转换字符串股票代码
    stockname(n)
    参数：n 整型
    返回：字符串
    '''
    s=str(n)                                #把数字转为字符串
    s=s.strip()                             #删除字符串前后空格
    if (len(s)<6 and len(s)>0):               #如果字符串长度为 1-5 的数，前面就用 0 补够 6 位长度
        s=s.zfill(6)+'.SZ'                 #深圳股后缀加.SZ
    if len(s)==6:                           #上海股票一般是 100000 以上的数字
        if s[0:1]=='0':                     #第一位为 0，是深圳股票代码
            s=s+'.SZ'                       #深圳股后缀加.SZ
        else:                               #否则是上海股票代码
            s=s+'.SH'                       #上海股后缀加.SH
    return s
print(stockname.__doc__)
print(stockname(776))
print(stockname(600030))
```

在 Python 中有一个奇妙的特性，即文档字符串“DocStrings”，用它可以为模块、类、函数等添加说明性的文字，使程序更易读易懂，更重要的是可以通过 Python 自

带的标准方法将这些描述性文字进行信息输出。

上面提到的自带的标准方法就是__doc__，需要注意，前后各有两个连在一起的下画线。

命令输出结果：

```
函数说明
数字股票代码转换字符串股票代码
stockname(n)
参数：n 整型
返回：字符串
000776.SZ
600030.SH
```

2. 函数命名规则

函数命名的规则同变量命名的规则，区分大小写，不能使用 Python 关键字作为函数名。

函数的命名一定要考虑其所完成的功能和语境，不要认为这是在浪费时间。当然，如果程序很短，又仅供自己使用，就无需考虑这些了。如果项目较大，而且文件较多，则最好花费些时间在函数命名上。这样一方面易于他人查看代码，另一方面便于自己维护。

为了更好地阅读程序，一般对函数命名有如下要求。

(1) 简单明了，即根据上下文给动词和介词加上名词，如使用 playMusic(file) 而不是 play(file)。不要为了过度的简洁而影响函数名称的清晰和准确性，也可以使用下画线连接词组的方式，如 set_volume(volume)，这样更方便通过函数名理解函数的功能。

(2) 避免歧义，即要考虑函数的命名是否存在多种解释，如在 displayName 中 display 是名词还是动词，不够清晰。如果命名不清晰，则需要重新命名消除歧义。再如，add 就可以使用 append 这类词语替换。

(3) 保持一致性，即在应用和库中使用相同的术语来描述同一个概念。避免在一个函数中使用 screen()，却在另外一个函数中使用 display()。

(4) 使用容易读懂的名称。最好是使用一段英文，或者英文短语，有时也可以使用拼音，最好不要中文和英文混合，这样读起来很费劲。

(5) 在必要时加前缀。前缀可以代表变量作用域的范围,也可以代表返回值的类型或分类。

例如:

```
def strGetUserId():
```

其中,前缀“str”表示返回值为字符串类型,不然别人会误以为返回整型数据。

3. 函数形式参数表

Python3 的函数定义非常简单,但灵活度却非常大。除了正常定义的必选参数外,还可以使用默认参数、可变参数和关键字参数,这就使得函数定义的接口不但可以处理复杂的参数,还可以简化调用者的代码。

函数参数的数据类型分为不可变数据和可变数据,其中不可变数据包括 Number (数字)、String (字符串)、Tuple (元组),可变数据包括 List (列表)、Dictionary (字典)、Set (集合)、对象数据。

不可变数据类型参数传递的只是值,没有影响传递参数变量的本身。

可变数据类型参数传递的是对象指针,如果修改了变量参数数据,传递参数的变量数据就会修改。因此,我们要对引入的数据进行复制备份并对数据副本进行处理,这样结果才是返回处理后的数据副本指针。

见示例 4-2。

```
#函数演示
i=20
f=19.86
dic = {i:2*i for i in range(10)}
l=[i for i in range(10)]
print('dic= ',dic)
print('l= ',l)

def fa(x,y):
    x=x*y
    y=x
    return y
def fb(x):
    x.clear()
    return x
```

```
def fc(x):
    y=x.copy() #使用 copy() 函数不会破坏原始数据
    y.clear()
    return y
a=fa(i,f)
print('a= ',a)
print('i= ',i)
print('f= ',f)
print('使用 copy() 函数，不会破坏原始数据。')
b=fc(dic)
print('b= ',b)
print('dic= ',dic)
c=fc(l)
print('c= ',c)
print('l= ',l)
print('不使用 copy() 函数，会破坏原始数据。')
d=fb(dic)
print('d= ',d)
print('dic= ',dic)
e=fb(l)
print('e= ',e)
print('l= ',l)
```

程序运行结果:

```
dic= {0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}
l= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
a= 397.2
i= 20
f= 19.86
使用 copy() 函数，不会破坏原始数据。
b= {}
dic= {0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}
c= []
l= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
不使用 copy() 函数，会破坏原始数据。
d= {}
dic= {}
e= []
l= []
```



1) 默认参数

默认参数是预先赋值，有初值的参数。默认参数可以简化函数的调用。在设置默认参数时，必选参数在前，默认参数在后，否则 Python 3 的解释器就会报错。

当函数有多个参数时，把变化大的参数放前面，变化小的参数放后面作为默认参数。

见示例 4-3。

```
#函数默认参数演示
def mypow(x,y=2):
    return x**y
a=mypow(4,3)
b=mypow(4)
print('a=mypow(4,3)= ',a)
print('b=mypow(4)= ',b)
```

程序运行结果：

```
a=mypow(4,3)= 64
b=mypow(4)= 16
```

2) 可变参数

可变参数就是在参数前面加了一个“*”号。在调用该函数时，就可以传入 0 个或任意个参数，这些可变参数会自动组装为一个元组（tuple）。

*args 是可变参数，args 接收的是一个元组。

元组无法直接修改，我们可以将它转换为一个列表，再来处理。请看一个利用可变参数来装配排序后列表的代码，见示例 4-4。

```
#函数*kargs 参数演示
def test(*k):
    return k

def shortall(*kargs):
    '''
    多参数字符串排序
    shortall(*kargs)
    返回字符串列表
    '''
```

```
#过滤非字符串
l=list(kargs).copy()
l.sort()
return l

s=test('ABC','222','1111','xixi','hahahah')
print('s= ',s)
print('type(s)= ',type(s))
#1.无参数
s1=shortall()
print('s1=',s1)
#2.字符串参数
s2=shortall('ABC','222','1111','xixi','hahahah')
print('s2=',s2)
#3.数值参数
s3=shortall(88,9,1,5,32423,999)
print('s3=',s3)
```

程序运行结果:

```
s= ('ABC', '222', '1111', 'xixi', 'hahahah')
type(s)= <class 'tuple'>
s1= []
s2= ['1111', '222', 'ABC', 'hahahah', 'xixi']
s3= [1, 5, 9, 88, 999, 32423]
```

3) 关键字参数

关键字参数允许传入 0 个或任意多个含参数名的参数，这些关键字参数在函数内部会自动组装为一个字典（dict）。

****kw** 是关键字参数，kw 接收的是一个字典。

见示例 4-5。

```
#函数**kargs 参数演示
def myset (**kw):
    '''
    **kw 是关键字参数
    mayset(**kw)
    返回参数内容
    '''
```

```

d=kw
return d
def detail(name=None,**kargs):
    '''
    detail(name=None,**kargs) -> str
    name is a str.return a str like'name,key1:value1,key2:value2'
    这个函数特定的功能
    '''
    data = []
    for x,y in kargs.items():
        data.extend(['', str(x), ':', str(y)])
    info = ''.join(data)
    return '%s%s'%(name,info)
a=myset(name='hhh',age=19,a=98.5,b=97,c=98.8)
print('a= ',a)
b=detail(name='hhh',age=19,a=98.5,b=97,c=98.8)
print('b= ',b)

```

程序运行结果:

```

a= {'name': 'hhh', 'age': 19, 'a': 98.5, 'b': 97, 'c': 98.8}
b= hhh,age:19,a:98.5,b:97,c:98.8

```

4) 参数组合

在 Python 中, 定义函数可以用必选参数、默认参数、可变参数和关键字参数, 这 4 种参数还可以一起使用, 或者只用其中几种。但是需要注意, 参数定义的顺序必须是必选参数、默认参数、可变参数和关键字参数。

使用函数参数的方式:

(1) 按位置匹配, 如 func(name)。

(2) 按关键字匹配, 如 func(key=value)。

(3) 按输入参数匹配, 如

元组收集: func(name,arg1,arg2)。

字典收集: func(name,key1=value1,key1=value2)。

(4) 按参数定义顺序。

4. 函数返回值

(1) `return` 语句的作用是结束函数调用并返回值。

`return` 语句可以不带参数也可以带多个参数。

不带参数就是结束函数运行，返回一个 `None` 作为返回值，类型是 `NoneType`。其与 `return`，`return None` 等效，都是返回 `None`。

带参数可以返回调用函数的语句，并且给变量表赋值。

一个函数中可以有任意多个 `return` 语句，其中任何一个 `return` 语句执行，该函数运行结束。

(2) 隐含返回值，指在整个函数体内没有 `return` 语句时，那么在函数运行结束时就会隐含返回一个 `None` 值作为返回值，类型是 `NoneType`。其与 `return`，`return None` 等效，都是返回 `None`。

请看示例 4-6。

```
#函数返回参数演示
#1.无 return 函数
def func1(x):
    x=x+1
a=func1(10)
print('a= ',a)
print('type(a)= ',type(a))
#2.有 return 函数，无返回参数
def func2(x):
    x=x+1
    return
#3.有 return 函数，返回参数 None
def func3(x):
    x=x+1
    return None
a=func1(10)
print('a= ',a)
print('type(a)= ',type(a))
b=func2(20)
print('b= ',b)
print('type(b)= ',type(b))
```



```

c=func3(40)
print('c= ',c)
print('type(c)= ',type(c))
#4. 多个返回参数
def func4(x):
    return x,x*2,x**2
d,e,f=func4(5)
print('d= ',d)
print('e= ',e)
print('f= ',f)

```

程序运行结果:

```

a= None
type(a)= <class 'NoneType'>
a= None
type(a)= <class 'NoneType'>
b= None
type(b)= <class 'NoneType'>
c= None
type(c)= <class 'NoneType'>
d= 5
e= 10
f= 25

```

5. 匿名函数

匿名函数表达式是基于数学中的 λ 演算而得名的,直接对应于其中的匿名函数表达式抽象,是一个匿名函数,即没有函数名的函数。

Python 也支持匿名函数,这些函数可以使用 λ (λ 是希腊字母中的第十一个字母,英文为 **lambda**) 关键字创建。

当我们在传入函数时,有些时候不需要显式地定义函数,直接传入匿名函数更方便。匿名函数只能有一个表达式,不用写 **return** 语句,返回值就是该表达式的结果。由于匿名函数没有名字,所以不必担心函数名冲突。此外,匿名函数也是一个函数对象,可以被赋值给一个变量,再利用变量来调用该函数。

见示例 4-7。

```
#1.lambda 匿名函数
f = lambda a,b,c:a+b+c
print( f(1,2,3))
#2.匿名函数及输入参数(lambda a,b=2:a*b)(i,j)
i=3
j=5
l=[x for x in range(1,(lambda a,b=2:a*b)(i,j))]
print(l)
#3.a = lambda *z:z      #*z 返回的是一个元组
f2 = lambda *z:z        #*z 返回的是一个元组
b=f2(1,3,6,9)
print('b= ',b)
#4.lambda **Arg: Arg   #arg 返回的是一个字典
f3=lambda **Arg: Arg    #arg 返回的是一个字典
c=f3(name='hhh',age=19,a=98.5,b=97,c=98.8)
print('c= ',c)
#5.lambda 嵌套到普通函数中, lambda 函数本身作为 return 的值
def addx(n):
    n+=n
    return (lambda x: n+n)(n)
d=addx(3)
print('d= ',d)
```

程序运行结果:

```
6
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
b= (1, 3, 6, 9)
c= {'name': 'hhh', 'age': 19, 'a': 98.5, 'b': 97, 'c': 98.8}
d= 12
```

6. __main__ 关键字

Python 3 用“if __name__ == '__main__':”语句说明程序的起始入口。

通过此语句可以判断:当前运行的模块是否是直接被 Python 解释器调用执行的,因为只有被 Python 解释器直接调用执行的模块文件才能获取程序参数。

getopt 是一个包装方法，用以读取 main 函数后面的参数。

getopt.getopt (args, options[, long_options]) 有如下三个参数：

args 就是 Python 命令行 “python demo_main.py -n 小白 -w 大家好!” 后面的参数 “demo_main.py -n 小白 -w 大家好!”，通常是 sys.argv 数组。不过我们一般会去除第一个元素，因为 sys.argv 的第一个元素就是文件名本身 “demo_main.py”，所以一般的写法是 sys.argv[1:]。

options 是一个字符串，描述了需要解析哪些参数。如果一个参数不需要参数值，就只写参数名，如 h；如果一个参数需要传入参数值，就在后面加冒号 “:”，如 n:。所以本案例中 hn:w: 的意思是有三个参数，分别是 -h、-n 和 -w，其中 -h 无需传入参数值，而 -n 和 -w 需要传入参数值。

long_options 是一个字符串数组，也表示需要解析哪些参数，它是相对 options 而言的。在 linux 中，我们经常会看到一个命令参数有多种写法，最常见的就是帮助参数。帮助参数有 -h 和 --help 两种写法，前一种是 options，后一种就是 long_options。

假如我们有一个 --help，那么在 long_options 中就是 ['help']。如果一个参数需要传入参数，如 --name 'Good' 在 long_options 中就是 ['name=']。是的，就是多了一个等于号 “=”。

getopt.getopt 返回一个元组 (opts, args)，其中 opts 就是我们解析出来的参数，而 args 则是剩余没有解析的参数。opts 是元组数组，每个元组就相当于一对键值 “key-value”，其中 key 就是我们的参数名（键），而 value 就是参数值。

见示例 4-8。

```
#main 函数示例
import getopt
import sys
def say(s1,s2):
    print( '你好，我叫', s1, ', ', s2)
if __name__ == '__main__':
    opts, args = getopt.getopt(sys.argv[1:], 'hn:w:', ['name=', 'word=',
        'help'])
    name = 'No Name'
    word = 'Hello'
    for key, value in opts:
        if key in ['-h', '--help']:
```

```
print( '一个向人打招呼的程序')
print( '参数: ')
print( '-h\t 显示帮助')
print( '-n\t 你的姓名')
print( '-w\t 想要说的话')

sys.exit(0)
if key in ['-n', '--name']:
    name = value
if key in ['-w', '--word']:
    word = value
say(name,word)
```

在 Spyder 中运行的结果:

```
你好, 我叫 No Name , Hello
```

因为程序中的变量 `name` 和 `word` 使用的是程序默认初值, 所以这个程序需要在 Windows 系统的 `cmd` 窗口来运行, 输入下面的命令。

```
Python demo_main.py -n 小白 -w 大家好!
```

运行结果如图 4-1。

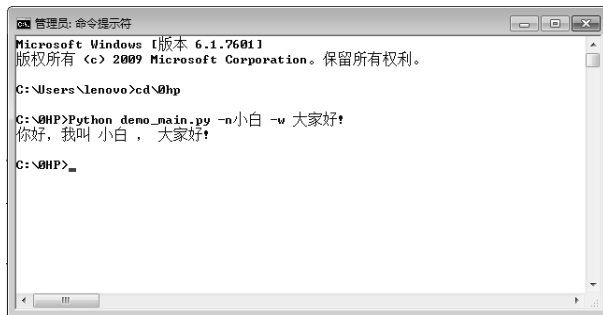


图 4-1 cmd 窗口的运行结果

4.2 Python 的类

面向对象的编程强调数据封装、继承等对象特性, 即对象=数据+方法(数据处理程序)。

在面向过程的编程中，def 语句是定义函数体。

Python 类中方法的定义也使用了 def 语句，所不同的是参数表不同，方法定义形式参数表中第一个参数永远是“self”关键字。

面向对象编程的基础是类的编程，下面介绍一些关于类的概念。

类 (Class): 用来描述具有相同的属性和方法的对象集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。

类变量: 类变量在整个实例化的对象中是公用的，它定义在类中且在方法体之外。

数据成员: 即类变量或者实例变量，用于处理类及其实例对象的相关数据。

方法: 类中定义的数据处理或操作，定义和使用类似于函数。

方法重写: 如果从父类继承的方法中不能满足子类的需求，就可以对其进行改写，这个过程叫作方法的覆盖 (override)，也称为方法的重写。

实例变量: 定义在方法中的变量，只作用于当前实例的类。

继承: 即一个派生类 (derived class) 继承基类 (base class) 的方法。继承也允许把一个派生类的对象作为一个基类的对象对待。

实例化: 创建一个类的实例，类的具体对象。类是抽象的模板，实例化是根据类创建出来的不同名字的具体“对象”。每个对象都拥有相同的方法，但各自的数据可能不同。

对象: 通过类定义的数据结构实例。对象包括两个数据成员 (类变量和实例变量) 和方法。

下面介绍 Python 中类的定义和使用。

1. 创建类

使用 class 语句创建一个新类，class 后面为类的名称并以冒号“:”结尾。

```
class ClassName:
    '类的帮助信息'          #类文档字符串
    class_suite              #类体
```

类的帮助信息可以通过“ClassName.__doc__”查看。class_suite 由类成员、方法、数据属性组成。

以下是一个简单的例子，示例 4-9 的部分代码。

```
class dog(object):                                #定义小狗类
    '这是一个 dog 类的定义。'
    def __init__(self,dogname='小虎'):           #类初始化
        self.name=dogname
        print('\n 你的小狗叫'+self.name+'.')

    def SetName(self,dogname):
        self.name=dogname
        print('你的小狗改名叫'+self.name+'.')

    def Run(self):
        print(self.name+'正在跑。')

    def CalltheDog(self):
        print('你呼唤\'\"'+self.name+'\",'+self.name+'向你跑来。\\n')
    def __del__(self):                             #类对象销毁
        class_name = self.__class__.__name__
        print( class_name+ " 销毁")
```

`__init__()`方法是一种特殊的方法，被称为类的构造函数或初始化方法。当创建了这个类的实例时，就会调用该方法。

`self` 代表类的实例。虽然在调用时不必传入相应的参数，但在定义类的方法时它是必须有的。

2. 实例化类对象

在其他编程语言中，实例化类一般用关键字 `new`，但是在 Python 中没有这个关键字，因此其类的实例化的方式类似于函数调用。

以下使用类的名称（`dog`）来实例化，并通过`__init__()`方法接收参数。示例 4-9 的部分代码。

```
#实例化第一个类 MyDoga
MyDoga=dog()                                     #使用默认参数
MyDoga.Run()
#实例化第一个类 MyDoga
MyDogb=dog('旺财')                               #使用自定义参数
MyDogb.CalltheDog()
```

```
MyDogb.SetName('小哈')
MyDogb.CalltheDog()
```

见示例 4-9。

#类演示代码

```
class dog(object):                                #定义小狗类
    '这是一个 dog 类的定义。'
    def __init__(self,dogname='小虎'):           #类初始化
        self.name=dogname
        print('\n 你的小狗叫'+self.name+'.')
    def SetName(self,dogname):
        self.name=dogname
        print('你的小狗改名叫'+self.name+'.')
    def Run(self):
        print(self.name+'正在跑。')
    def CalltheDog(self):
        print('你呼唤\"'+self.name+'\",'+self.name+'向你跑来。\\n')
    def __del__(self):                             #类对象销毁
        class_name = self.__class__.__name__
        print( class_name+ " 销毁")
#实例化第一个类 MyDoga
MyDoga=dog()                                       #使用默认参数
MyDoga.Run()
#实例化第一个类 MyDoga
MyDogb=dog('旺财')                               #使用自定义参数
MyDogb.CalltheDog()
MyDogb.SetName('小哈')
MyDogb.CalltheDog()
```

程序运行结果:

```
你的小狗叫小虎。
小虎正在跑。
你的小狗叫旺财。
你呼唤"旺财",旺财向你跑来。
你的小狗改名叫小哈。
你呼唤"小哈",小哈向你跑来。
```

3. 访问属性

可以使用点号 “.” 来访问对象的属性。

在上例中，可以通过 SetName()方法来修改 dog 的名字，也可以直接修改类属性。

见示例 4-10。

```
class dog(object):                                #定义小狗类
    '这是一个 dog 类的定义。'
    def __init__(self,dogname='小虎'):            #类初始化
        self.name=dogname
        print('\n 你的小狗叫'+self.name+'.')
    def SetName(self,dogname):
        self.name=dogname
        print('你的小狗改名叫'+self.name+'.')
    def Run(self):
        print(self.name+'正在跑。')
    def CalltheDog(self):
        print('你呼唤\'\"'+self.name+'\",'+self.name+'向你跑来。\\n')
    def __del__(self):                             #类对象销毁
        class_name = self.__class__.__name__
        print( class_name+ " 销毁")

#实例化第一个类 MyDoga
MyDogc=dog('小黑')                                #使用自定义参数
MyDogc.CalltheDog()
MyDogc.name="旺旺财"
MyDogc.CalltheDog()
```

程序运行结果：

```
你的小狗叫小黑。
你呼唤"小黑",小黑向你跑来。
你呼唤"旺旺财",旺旺财向你跑来。
```

4. 类的内置属性

Python 类的内置属性的种类及含义见表 4-1。

表 4-1 Python 类的内置属性

| 类属性 | 含义 |
|-------------------------|--|
| <code>__name__</code> | 类的名字（字符串） |
| <code>__doc__</code> | 类的文档字符串 |
| <code>__bases__</code> | 类的所有父类组成的元组 |
| <code>__dict__</code> | 类的属性组成的字典，包含一个字典，由类的数据属性组成 |
| <code>__module__</code> | 类所属的模块。类的全名是 <code>__main__.className</code> ，如果类位于一个导入模块 <code>mymod</code> 中，那么 <code>className.__module__</code> 就等于 <code>mymod</code> |
| <code>__class__</code> | 类对象的类型 |

见示例 4-11。

```

class dog(object):                                #定义小狗类
    '这是一个 dog 类的定义。'
    def __init__(self,dogname='小虎'):            #类初始化
        self.name=dogname
        print('\n 你的小狗叫'+self.name+'.')
    def SetName(self,dogname):
        self.name=dogname
        print('你的小狗改名叫'+self.name+'.')
    def Run(self):
        print(self.name+'正在跑。')
    def CalltheDog(self):
        print('你呼唤\'"+self.name+"\','+self.name+'向你跑来。')
    def __del__(self):    #类对象销毁
        class_name = self.__class__.__name__
        print( class_name+ " 销毁")

print('dog.__name__ : ',dog.__name__)
print('dog.__doc__ : ',dog.__doc__)
print('dog.__bases__ : ',dog.__bases__)
print('dog.__dict__ : ',dog.__dict__)
print('dog.__module__ : ',dog.__module__)
print('dog.__class__ : ',dog.__class__)

```

程序运行结果：

```
dog.__name__ : dog
```

```
dog.__doc__ : 这是一个 dog 类的定义。
dog.__bases__ : (<class 'object'>,)
dog.__dict__ : {'__module__': '__main__', '__doc__': '这是一个 dog 类的定义。', '__init__': <function dog.__init__ at 0x0000000009C30048>, 'SetName': <function dog.SetName at 0x0000000009C30A60>, 'Run': <function dog.Run at 0x0000000009C30B70>, 'CalltheDog': <function dog.CalltheDog at 0x0000000009C301E0>, '__del__': <function dog.__del__ at 0x0000000009C30D08>, '__dict__': <attribute '__dict__' of 'dog' objects>, '__weakref__': <attribute '__weakref__' of 'dog' objects>}}
dog.__module__ : __main__
dog.__class__ : <class 'type'>dog.__module__ : __main__
```

5. 类对象销毁

Python 使用了引用计数来跟踪和回收类对象。当类对象（类实例）被创建时，就已经创建了一个引用计数。当类对象不再使用，即这个对象的引用计数变为 0 时，它就要被销毁。但是销毁不是立即操作的，而是由 Python 解释器在适当的时机进行，并回收类对象占用的内存空间。用户也可以在程序中主动销毁类对象。

Python 类有析构函数“__del__”。它在对象销毁时被调用，并且当对象不再被使用时，用户可以使用 del 语句销毁类对象。

```
del MyDoga
del MyDogb
del MyDogc
```

见示例 4-12。

```
class dog(object):                                #定义小狗类
    '这是一个 dog 类的定义。'
    def __init__(self,dogname='小虎'): #类初始化
        self.name=dogname
        print('\n 你的小狗叫'+self.name+'.')
    def SetName(self,dogname):
        self.name=dogname
        print('你的小狗改名叫'+self.name+'.')
    def Run(self):
        print(self.name+'正在跑。')
```

```

def CalltheDog(self):
    print('你呼唤\'"+self.name+"\','+self.name+'向你跑来。\\n')
def __del__(self):
    #类对象销毁
    class_name = self.__class__.__name__
    print( class_name+ " 销毁")

#实例化第 1 个类 MyDoga
MyDoga=dog()
MyDoga.CalltheDog()
#实例化第 2 个类 MyDoga
MyDogb=dog('旺财')
MyDogb.CalltheDog()
#实例化第 3 个类 MyDoga
MyDogc=dog('小黑')
MyDogc.CalltheDog()
del MyDoga
del MyDogb
del MyDogc
MyDogc.CalltheDog()

```

#使用默认参数

#使用自定义参数

#使用自定义参数

#类示例销毁，再执行此句会出错

程序运行结果:

```

你的小狗叫小虎。
你呼唤"小虎",小虎向你跑来。
你的小狗叫旺财。
你呼唤"旺财",旺财向你跑来。
你的小狗叫小黑。
你呼唤"小黑",小黑向你跑来。
dog 销毁
dog 销毁
dog 销毁
MyDogc.CalltheDog()
NameError: name 'MyDogc' is not defined

```

6. 类的继承

面向对象的编程的主要好处之一是代码的重用，而实现重用的方法之一是通过继承机制。

通过继承创建的新类称为子类或派生类，被继承的类称为基类、父类或超类。
继承语法：

```
class 派生类名(基类名)
...
```

在 Python 中继承的一些特点：

(1) 如果在子类中需要直接使用父类的构造方法，就需要显式调用或者不重写父类的构造方法。

(2) 在调用基类的方法时，需要加上基类的类名前缀，且需要带上 `self` 参数变量。而在类中调用普通方法时不需要带上 `self` 参数。

(3) Python 会先在派生类中查找对应类型的方法，如果找不到，就会到基类中逐个查找。

如果在继承元组中罗列了一个以上的类，那么它就被称作多重继承。

派生类的声明与它们的父类类似，继承的基类列表跟在类名之后，如下所示：

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
...
```

7. 方法重写

如果父类方法的功能不能满足需求，就可以在子类重写父类的方法。

下面给出有关类继承和方法重新的演示代码，见示例 4-13。

```
class dog(object):                                #定义小狗类
    '这是一个 dog 类的定义。'
    def __init__(self,dogname='小虎'):            #类初始化
        self.name=dogname
        print('\n 你的小狗叫'+self.name+'.')
    def SetName(self,dogname):
        self.name=dogname
        print('你的小狗改名叫'+self.name+'.')
    def Run(self):
        print(self.name+'正在跑。')
    def CalltheDog(self):
        print('你呼唤\"'+self.name+'\",'+self.name+'向你跑来。\\n')
    def __del__(self):
```

```

class_name = self.__class__.__name__
print( class_name+ " 销毁")

class newdog(dog):
    '这是新 dog 类'
    def Eat(self):                                #定义子类新方法
        print(self.name+'正在吃东西。')
    def CalltheDog(self):                        #重构基类方法
        print('你呼唤\'"+self.name+"\','+self.name+'抬头看你。\\n')

#实例化第一个类 MyDoga
MyDoga=newdog()                                #使用默认参数
MyDoga.Run()
MyDoga.Eat()
print('-----')
MyDoga.name='宝贝'
MyDoga.Run()
MyDoga.Eat()
MyDoga.CalltheDog()
del MyDoga

```

程序运行结果:

```

你的小狗叫小虎。
小虎正在跑。
小虎正在吃东西。

-----
宝贝正在跑。
宝贝正在吃东西。
你呼唤"宝贝",宝贝抬头看你。
newdog 销毁

```

8. 类基础方法重载

类能重载的基础方法的种类及功能, 见表 4-2。

表 4-2 类基础方法重载

| 方法 | 描述及简单的调用 |
|---|--|
| <code>__init__(self [,args...])</code> | 构造方法。 简单的调用方法: <code>obj = className(args)</code> |
| <code>__del__(self)</code> | 析构函数, 删除一个对象。 简单的调用方法: <code>del obj</code> |
| <code>__repr__(self)</code> | 转化为供解释器读取的形式。 简单的调用方法: <code>repr(obj)</code> |
| <code>__str__(self)</code> | 用于将值转换为适于文本阅读的形式。 简单的调用方法: <code>str(obj)</code> |
| <code>__cmp__(self, x)</code> | 对象比较。 简单的调用方法: <code>cmp(obj, x)</code> |

Python 除了支持基础方法重载, 同样支持运算符重载, 见表 4-3。

表 4-3 常用运算符重载方法

| 方法 | 说明 |
|---------------------------|-------------|
| <code>__add__</code> | 加法 (+) |
| <code>__sub__</code> | 减法 (-) |
| <code>__mul__</code> | 乘法 (*) |
| <code>__truediv__</code> | 除法 (/) |
| <code>__floordiv__</code> | 整除 (//) |
| <code>__mod__</code> | 取模 (求余) (%) |
| <code>__pow__</code> | 幂 (**) |

9. 类的 `super()` 方法

在类的继承中, 如果重定义某个方法, 子类方法就会覆盖父类的同名方法。但如果我们希望能同时使用父类 (基类) 的同名方法, 就可以通过 `super()` 方法来实现。

`super()` 方法是用于调用父类 (基类) 的一个方法。

`super()` 方法是用来解决多重继承问题的。在使用单继承时, 可以直接用类名调用父类方法, 这时如果使用多继承就会涉及查找顺序 (MRO)、重复调用 (钻石继承) 等问题。

`super()`方法的语法:

```
super(type[, object-or-type])
```

`super()`方法的参数见表 4-4。

表 4-4 `super()`方法的参数

| 参数 | 说明 |
|----------------|------------------|
| type | 类名 |
| object-or-type | 类实例或类名, 一般是 self |

事实上, 对于定义的每一个类, Python 都会计算出一个方法解析顺序 (Method Resolution Order, MRO) 列表, 它代表了类继承的顺序。一个类的 MRO 列表就是合并所有父类的 MRO 列表, 并遵循以下三条原则。

- (1) 子类永远在父类前面。
- (2) 如果有多个父类, 就会根据它们在列表中的顺序被检查。
- (3) 如果下一个类存在两个合法的选择, 就选择第一个父类。

`super()`方法的使用, 见示例 4-14。

```
class A(object):
    def __init__(self):
        print("init A Class")
        super(A, self).__init__()
class B(object):
    def __init__(self):
        print("init B Class")
        super(B, self).__init__()
class C(object):
    def __init__(self):
        print("init C Class")
        super(C, self).__init__()
class D(A,B,C):
    def __init__(self):
        print("init D class")
        super(D, self).__init__()
class E(D):
    def __init__(self):
```

```
        print("init E class")
        super(E, self).__init__()
class F(E):
    def __init__(self):
        print( "init F class")
        super(F, self).__init__()

F = F()
```

程序运行结果:

```
init F class
init E class
init D class
init A Class
init B Class
init C Class
```

10. 类的属性与方法

类的私有属性如下:

__private_attrs: 以两个连接的下画线开头, 声明该属性为私有, 不能在类的外部使用或直接访问。在类的内部使用时为 `self.__private_attrs`。

类的方法: 在类的内部使用 `def` 关键字可以为类定义一个方法。与一般函数定义不同, 类方法必须包含参数 `self`, 且为第一个参数。

类的私有方法如下:

__private_method: 以两个连接的下画线开头, 声明该方法为私有方法, 不能在类的外部调用。在类的内部调用时为 `self.__private_methods` (`self.__私有方法`)。

Python 不允许类实例访问私有属性, 但可以使用 `object._className__attrName` (对象名.类名私有属性名) 访问私有属性。见示例 4-16 的 `print(t._testPrivate__data)`。

单下画线、双下画线、头尾双下画线的说明 (foo 为名字, 可以是变量名或方法名):

_foo: 以单下画线开头表示的是 `protected` 类型的变量, 即保护类型, 只允许其本身与子类进行访问, 不能用于 `from module import *`。

`__foo`: 以两个连接的下画线开头表示的是私有类型 `private` 的变量, 只允许这个类进行访问。

`__foo__`: 其定义的是特殊方法, 一般是系统定义名字, 类似于 `__init__()`。见示例 4-15。

```
class testPrivate:
    def __init__(self):
        self.__data = []

    def add(self,item):
        self.__data.append(item)
    def printData(self):
        print (self.__data)

t = testPrivate()
t.add('dancingrain')
t.add('hello')
t.printData()
print(t.__data)
```

程序运行结果:

```
['dancingrain', 'hello']
print(t.__data)
AttributeError: 'testPrivate' object has no attribute '__data'
```

在程序中, 如果直接访问私有变量就会出现错误提示“`AttributeError: 'testPrivate' object has no attribute '__data'`”。

但是, 这并不意味着我们就不能从外部访问这个变量了。前面说过 Python 在类的内部用 `_classname__foo` 替换了 `__foo`, 因此, 我们可以在类的外面使用 `_testPrivate__data` 访问 `__data`。代码见示例 4-16。

```
class testPrivate:
    def __init__(self):
        self.__data = []
    def add(self,item):
        self.__data.append(item)
    def printData(self):
        print (self.__data)
```

```
t = testPrivate()
t.add('testPrivate Var')
t.add('hello')
t.printData()
print(t._testPrivate__data)
```

程序运行结果:

```
['testPrivate Var', 'hello']
['testPrivate Var', 'hello']
```

上面结果表明, 我们获取到了私有变量的值。

使用私有变量是为了防止其他程序员或用户误改关键数据。如果一定要修改内部私有数据, 就使用类内部方法间接修改。

4.3 Python的作用域

作用域就是能够起作用的范围。在 Python 中, 并不是所有的语句块都会产生作用域, 只有当变量在模块、类、函数中定义的时候才会有作用域的概念, 因此变量、函数和类都有作用域。

1. 块级作用域

块级作用域指在最小单元中的局部语句作用域。见示例 4-17。

```
#1. 块级作用域
s1='s1 是块级外定义的变量。'

print(s1)
if 1>0:
    s = "if 块中定义的变量 s"
    s1='s1 是块级内定义的变量。'

print(s)
print(s1)
for i in range(10):
    if i==2:
        m = i
```

```
print('for 语句中定义的 i= ',i)
print('for 语句中的 if 语句块定义的 m= ',m)
```

程序运行结果:

```
s1 是块级外定义的变量。
if 块中定义的变量 s
s1 是块级内定义的变量。
for 语句中定义的 i= 9
for 语句中的 if 语句块定义的 m= 2
```

在示例 4-17 中, 变量 s 是在 if 语句中定义的, 但是在 if 语句外可以使用。

变量 s1 在 if 语句外定义, 但是在 if 语句内部却把变量 s1 修改了。

for 语句定义了子变量 i, for 语句内的 if 语句定义了变量 m, 那么在 for 语句外就能输出或使用变量 i 和变量 m。

从上面的结果中看到, 变量的定义和赋值都是同一个作用域的同一个块在 if-elif-else, for-else, while, try-except, try-finally (其中不能含 def 语句和 class 语句) 等关键字的语句块中并不会产生新的作用域。因此, 块作用域中的变量在整个块中有效, 其他语句可以创建、使用或修改这些变量。

2. 函数级作用域

函数级作用域是指最小函数体的作用域。在 Python 3 中, 因为函数体内可以定义子函数, 因此这里指无定义子函数的函数体。见示例 4-18。

```
#2. 函数体中的作用域
s1='s1 是块级定义的变量。'
def func():
    s1='s1 是函数内定义的变量。'
    print('函数内输出 s1:'+s1)
func()          #执行函数
print('函数外输出 s1:'+s1)
```

程序运行结果:

```
函数内输出 s1:s1 是函数内定义的变量。
函数外输出 s1:s1 是块级定义的变量。
```

从运行结果中我们可以看到, 在函数体外定义了一个变量 s1, 函数体内对变量

s1 进行赋值，并显示结果为“s1 是函数内定义的变量”。

函数运行结束再次输出变量 s1 的值为“s1 是块级定义的变量”，这说明函数体外的变量 s1 和函数体内的变量 s1 不相同，因此它们不在同一个作用域。

再看一个程序，示例 4-19。

```
#3.函数体中的作用域
s1='s1 是块级定义的变量。'
def func():
    print('函数内输出 s1: '+s1)
func()           #执行函数
print('函数外输出 s1: '+s1)
```

程序运行结果：

```
函数内输出 s1: s1 是块级定义的变量。
函数外输出 s1: s1 是块级定义的变量。
```

上面结果说明，函数体可以使用其外部定义的变量 s1，但是不能修改。

再看示例 4-20，如果在函数体内定义变量 s2，那么在函数体外使用会是什么结果？

```
#4.函数体中的作用域
def func():
    s2='s2 是函数级定义的变量。'
    print('函数内输出 s2: '+s2)
func()           #执行函数
print('函数外输出 s2: '+s2)
```

程序运行结果：

```
print('函数外输出 s2: '+s2)

NameError: name 's2' is not defined)
```

程序运行出错，提示变量 s2 没有定义。这说明函数体内定义的变量仅仅在函数体内有效，函数运行结束，变量消失，后面再想使用这些变量就会出现错误。

由此可见，函数体内的变量和函数体外的变量不在一个作用域。

3. 作用域的类型

在 Python 中，当使用一个变量时并不严格要求预先声明，但必须绑定到当前作

用域中。

Python 的作用域分为 L (Local、局部) 作用域、E (Enclosing、嵌套) 作用域、G (Global、全局) 作用域和 B (Built-in、内置) 作用域。

L 作用域是在 Python 函数体内创建使用的变量或定义的函数体，因为只有当前函数和函数体中定义的子函数才能够访问。由于 Python 没有 Local 关键字，因此没有用 Global 定义的变量都是局部变量。Python 中也有递归，即自己调用自己，每次调用都会创建一个新的局部命名空间。在函数内部的变量声明中，使用 Global 关键字的为全局变量，其他均为局部变量。

E 作用域包含在 def 关键字闭包函数中。E 作用域和 L 作用域是相对的，即 E 作用域是相对于更上层的函数 L 作用域而言的。但区别在于，对于一个函数，L 作用域是定义在此函数内部的局部作用域，而 E 作用域是定义在此函数的上一层父级函数的局部作用域，其主要是为了实现 Python 的闭包而增加的。

G 作用域是在 Python 当前模块文件中代码运行时就创建的变量。在当前模块的所有函数都能引用或访问的变量为全局变量，即定义为全局作用域变量名，其在 Python 当前文件中任意位置都能使用。在当前文件任意位置中用 Global 语句定义的变量，文件中每一个模块都能使用这些变量。建议尽量少定义全局变量，因为全局变量在模块文件运行的过程中会一直存在，占用内存空间。

B 作用域：即系统内固定模块里定义的变量，如预定义在 Built-in 模块内的变量。

4. 变量名解析 LEGB 法则

LEGB 法则：以 L, E, G, B 的规则查找，即在局部找不到，就去局部外的局部找（如闭包），再找不到就去全局找，然后再去内建中找。

当在函数中使用未确定的变量名时，Python 会按照优先级依次搜索 4 个作用域，以此来确定该变量名的意义。首先搜索局部作用域，然后是上一层嵌套结构中 def 函数或 lambda 函数的嵌套作用域，接着是全局作用域，最后是内置作用域，即搜索变量名的优先级为局部作用域 > 嵌套作用域 > 全局作用域 > 内置作用域。

在 Python 3 解释器查找变量时，会按照顺序依次查找，直到找到变量则停止查找，但如果查找完没有找到对应的变量，则抛出 “NameError: name 'xxxx' is not defined” 的异常。

下面这个例子比示例 4-20 仅仅在函数体中多了一条 “global s 2”。见示例 4-21。

#5. 函数体中的作用域



```
def func():
    global s2
    s2='s2 是函数级定义的变量。'
    print('函数内输出 s2: '+s2)
func()          #执行函数
print('函数外输出 s2: '+s2)
```

程序运行结果:

```
函数内输出 s2: s2 是函数级定义的变量。
函数外输出 s2: s2 是函数级定义的变量。
```

从上面的显示结果中看到, 在函数体中使用 **Global** 语句, 其后面的变量为全局变量。模块文件.py 中的 **Global** 变量在该模块文件中都存在作用域。

5. 命名空间

命名空间是为了防止不同的人在编写类和库时发生命名冲突而设计的。命名空间可以使变量、函数、类的名称作用在本空间内, 而其他空间仍可以使用同样的名称。就好比不同的文件夹下可以有相同的文件名一样, 但在相同的文件夹下不能有相同的文件名, 命名空间就相当于一个虚拟的文件夹。

Python 3 的 LEGB 法则仅仅适合同一个模块的变量的作用域, 不同文件的作用域是不相同的。因此, 我们可以使用模块名或模块的别名作为命名空间, 这样就可以实现不同模块之间的变量数据共享了。

一般我们开发的系统都是由很多模块文件构成的。例如, 后面给大家介绍的小白量化分析系统就是由很多模块文件构成的, 不同模块可以共享或者修改系统默认参数变量。

小白量化投资系统模块的全局变量都在 **HP_global.py** 模块文件中, 其中模块文件 **HP_set.py** 对它的变量进行初始化赋值。其他模块文件, 只要是包含 **import HP_global as g** 语句的就能获取命名空间“g”的所有全局变量值, 也可以进行修改这些变量。

见示例 4-22。

```
import HP_global as g
'''
#-----
```

下面是 HP_global 的部分内容

#运行系统环境设置

#os=1 windows,=2 linux, =3 mac os

global os #操作系统

global pyver #Python 版本

#软件名称设置

global name #软件名称

global title #软件标题

global developer #软件开发者

global ver #软件版本号

#软件数据目录

global datapath #数据目录

global prgpath #软件目录

#-----

'''

import HP_set

'''

#-----

下面是 HP_set 的部分内容

##数据主目录

g.datapath='\\xbdata'

g.prgpath='\\xb'

#软件名称

g.root=None

g.name='小白量化投资平台'

g.title='小白量化投资平台(1.00 版)'

g.ico='tt.ico'

g.winW=1200

g.winH=760

g.ver=1.01

g.user='18578755056'

g.login=False

g.os=1

#-----

'''

#g.name 是 g 命令空间的全局变量, 整个运行期间所有模块都能访问和修改

#我们利用这个模块来获取这些定制的值

```
print('软件名称: ',g.name)
print('软件标题: ',g.title)
print('下面修改 g 命名空间变量')
g.name='荷蒲证券分析研究平台'
g.title='荷蒲证券分析研究平台(1.00 版)'
print('软件名称: ',g.name)
print('软件标题: ',g.title)
```

程序运行结果:

```
软件名称:  小白量化投资平台
软件标题:  小白量化投资平台(1.00 版)
下面修改 g 命名空间变量
软件名称:  荷蒲证券分析研究平台
软件标题:  荷蒲证券分析研究平台(1.00 版)
```

从上面的结果中看到, 示例 4-22 中的程序修改了命名空间“g”的全局变量。因此, 在运行的程序中任何一个模块文件都能获取或修改命名空间“g”的全局变量。



5

第 5 章

NumPy 库与多维数组

5.1 NumPy的简介

NumPy 是应用 Python 进行科学计算的基础库。它是一个由多维数组对象、多种衍生的对象（如掩码式数组“masked arrays”或矩阵），以及一系列为快速计算数组而生的例程，包括数学运算、逻辑运算、形状操作、排序、选择、I/O、离散傅里叶变换、基本线性代数、基本统计运算、随机模拟等组成的库。

NumPy 提供了高级的数值编程工具和精密的运算库，它已经在很多大型金融公司、科学计算组织中得到广泛使用。

NumPy 数组和标准 Python 序列的不同之处：

（1）NumPy 数组在创建时就已经固定了尺寸，当尺寸发生改变时就会创建一个新的数组。而 Python 中的列表数据类型可以动态变化。

（2）在同一个 NumPy 数组中所有元素的数据类型都要一致，并在内存中占有相同的大小。但 Python 中的列表可包含不同数据类型的元素。

（3）在数据量巨大时，使用 NumPy 进行高级数据运算比使用 Python 的内置序列更有效，执行代码也更少。

（4）在 Python 中许多科学计算库是建立在 NumPy 库基础上的，虽然这些科学计算库也保留了 Python 内置序列的输入接口，但实际上在输入前还是要转成 NumPy 数组，它们的计算结果输出一般也是 NumPy 数组。

（5）NumPy 数组运算比 Python 中的列表数据类型运算更简洁，运算速度也更快。



当 Python 中的列表数据“lista”与“listb”相乘时，需要使用 for 循环语句才能实现，而且当列表长度增加时，运算速度很慢。而当 NumPy 的数组“na”与数组“nb”相乘时，只需要一条命令“na*nb”即可实现，且运算速度非常快。

因此，在数值运算、仿真、深度学习中会经常使用 NumPy 库中的函数，并且 NumPy 通常与 SciPy（Scientific Python）和 Matplotlib（绘图库）一起使用。

5.2 NumPy库的安装和使用

在 Linux 和 macOS 系统下的安装：

```
sudo pip install numpy
```

在 Windows 系统下的安装：

```
pip install numpy
```

在 Python 中，引入 NumPy 库使用命令：

```
import numpy
```

通常使用别名“np”，即引入 NumPy 库使用命令：

```
import numpy as np
```

1. ndarray 数组属性

在 NumPy 库中，最重要的对象是被称为 ndarray 的 N 维数组类型。它描述的是相同类型的元素集合，可以使用基于零的索引访问该集合中的项目。

ndarray 数组的对象属性及说明见表 5-1。

表 5-1 ndarray 数组属性

| ndarray 属性 | 说明 |
|---------------|--|
| ndarray.ndim | 数组轴的个数，在 Python 中轴的个数被称作秩 |
| ndarray.shape | 数组的维度，这是一个指数组在每个维度上大小的整数元组。例如，一个 n 排 m 列的矩阵，它的 shape 属性是 (2,3)，这个元组的长度显然是秩，即维度或者 ndim 属性 |
| ndarray.size | 数组元素的总个数，等于 shape 属性中元组元素的乘积 |
| ndarray.dtype | 一个用来描述数组中元素类型的对象，可以通过创造或指定 dtype 使用标准 Python 类型。另外，NumPy 提供自己的数据类型 |

续表

| ndarray 属性 | 说明 |
|------------------|---|
| ndarray.itemsize | 数组中每个元素的字节大小。例如，一个元素类型为 float64 的数组，其 itemsize 属性值为 8（64/8=8）。又如，一个元素类型为 complex32 的数组，其 item 属性为 4（32/8=4） |
| ndarray.data | 包含实际数组元素的缓冲区，通常我们不需要使用这个属性，因为我们是通过索引来使用数组中的元素的 |

2. ndarray 数组的创建

ndarray 中的每个元素在内存中使用相同大小的块，是数据类型的对象，称为 dtype。

基本的 ndarray 是使用 NumPy 中的数组函数创建的，创建格式如下：

```
numpy.array(object, dtype = None, copy = True, order = None, subok = False,
ndmin = 0)
```

ndarray 数组参数的种类及描述见表 5-2。

表 5-2 ndarray 数组的参数

| 参数 | 描述 |
|--------|--|
| object | 任何暴露数组接口方法的对象都会返回一个数组或任何（嵌套）序列 |
| dtype | 数组的所需数据类型，可选 |
| copy | 可选，默认为 True，对象是否被复制 |
| order | C（按行）、F（按列）或 A（任意，默认） |
| subok | 在默认（False）情况下，返回的数组被强制为基类数组。如果为 True，则子类就被传递 |
| ndimin | 指定返回数组的最小维数 |

NumPy 中能够创建 ndarray 数组的函数及功能见表 5-3。

表 5-3 创建 ndarray 数组的函数

| 函数 | 功能 |
|------------------|--|
| numpy.arange() | numpy.arange([start,] stop[, step,], dtype=None)，该函数返回的是一个均匀分布的数组 |
| numpy.linspace() | numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)是通过指定起始值、终止值、元素个数来创建一个等差数组，默认为前闭后闭，通过设置 endpoint=False 可以对终止值设置是否包含 |
| numpy.logspace() | numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)返回等比数组，默认以 10 为底，通过设置 base 改变底值 |

续表

| 函数 | 功能 |
|---------------------------------|--|
| <code>numpy.zeros()</code> | <code>numpy.zeros(shape, dtype=float, order='C')</code> 返回一个给定形状和类型的用 0 填充的数组 |
| <code>numpy.ones()</code> | <code>numpy.ones(shape, dtype=None, order='C')</code> 返回一个给定形状和类型的并用 1 填充的数组 |
| <code>numpy.eye()</code> | <code>numpy.eye(N, M=None, k=0, dtype=<type 'float'>)</code> 返回一个二维数组 (N,M)，对角线的地方为 1，其余的地方为 0 |
| <code>numpy.identity()</code> | <code>numpy.identity(n, dtype=None)</code> 返回一个方阵，即标准意义的单位阵 |
| <code>numpy.empty()</code> | <code>numpy.empty(shape, dtype=float, order='C')</code> 依据给定形状和类型返回一个新的空数组 |
| <code>numpy.fromstring()</code> | <code>numpy.fromstring(string, dtype=float, count=-1, sep='')</code> 通过字符串来创建数组，字符串的每一个字符占用一个字节的空間，也就是 8 个字节，通过指定 <code>dtype</code> 创建不同的数组。当 <code>dtype=int8</code> 时，每个字符对应一个元素；当 <code>dtype=16</code> 时，两个字符对应一个元素，并且以十六进制保存（高位的字符需要乘以 16） |
| <code>numpy.asarray()</code> | <code>numpy.asarray(a, dtype=None, order=None)</code> 从现有数据中创建数组。a 为任意形式的输入参数，比如列表、列表的元组、元组、元组的元组、元组的列表 |
| <code>numpy.reshape()</code> | <code>numpy.reshape(a, newshape, order='C')</code> 调整数组形状 |

示例 5-1 给出了创建数组的演示。

```
import numpy
print( '使用列表生成一维数组 numpy.array([1,2,3,4,5,6]) ' )
x = numpy.array([1,2,3,4,5,6])
print( x )                                #打印数组
print( 'x.dtype : ',x.dtype )             #打印数组元素的类型
print( '使用列表生成二维数组 x = numpy.array([[1,2],[3,4],[5,6]]) ' )
x = numpy.array([[1,2],[3,4],[5,6]])
print( x )                                #打印数组
print('x.ndim', x.ndim)                   #打印数组的维度
print('x.shape', x.shape)                 #打印数组各个维度的长度，shape 是一个元组
print( 'numpy.zeros(6)' )
x = numpy.zeros(6)                         #创建一维长度为 6 的一维 0 数组
print( x )
print( 'numpy.zeros((2,3))' )
x = numpy.zeros((2,3))                    #创建一维长度为 2，二维长度为 3 的二维 0 数组
print( x )
print( 'numpy.ones((2,3))' )
x = numpy.ones((2,3))                    #创建一维长度为 2，二维长度为 3 的二维 1 数组
print( x )
print( 'numpy.empty((3,3)) ' )
```

```

x = numpy.empty((3,3)) #创建一维长度为 2，二维长度为 3，未初始化的二维数组
print( x)
print( '使用 arrange 生成连续元素')
print( numpy.arange(6))           #[0,1,2,3,4,5,] 开区间
print( numpy.arange(0,6,2))       #[0, 2, 4]
print( 'numpy.eye(3)')
print(numpy.eye(3))

```

程序运行结果:

```

使用列表生成一维数组 numpy.array([1,2,3,4,5,6])
[1 2 3 4 5 6]
x.dtype : int32
使用列表生成二维数组 x = numpy.array([1,2],[3,4],[5,6]])
[[1 2]
 [3 4]
 [5 6]]
x.ndim 2
x.shape (3, 2)
numpy.zeros(6)
[0. 0. 0. 0. 0. 0.]
numpy.zeros((2,3))
[[0. 0. 0.]
 [0. 0. 0.]]
numpy.ones((2,3))
[[1. 1. 1.]
 [1. 1. 1.]]
numpy.empty((3,3))
[[3.55727265e-322 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 4.74303020e-321]
 [8.06632138e-308 1.20160711e-306 1.69119330e-306]]
使用 arrange 生成连续元素
[0 1 2 3 4 5]
[0 2 4]
numpy.eye(3)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

```

5.3 ndarray 数组元素的数据类型

ndarray 数组元素的数据类型及说明见表 5-4。

表 5-4 ndarray 数组元素的数据类型

| 数据类型 | 说明 |
|----------------------------------|---|
| int8/int16/int32/int64 | 有符号 8/16/32/64 位整数，类型代码 i1/i2/i4/i8，分别为 1/2/4/8 个字节 |
| uint8/uint16/uint32/uint64 | 无符号 8/16/32/64 位整数，类型代码 u1/u2/u4/u8 |
| float16/float32/float64/float128 | 浮点数，类型代码 f2/f4/f8/fl6 |
| complex64/complex128/complex256 | 复数，类型代码 c8/c16/c32 |
| bool | 布尔类型，类型代码 ? |
| object | 对象类型，类型代码 O |
| string_ | 固定长度的字符串类型（每个字符一个字节），类型代码 S。例如，要创建一个长度为 10 的字符串，应使用 S10 |
| unicode | 固定长度的 unicode 类型，类型代码 U，同字符串，如 U10 |

数据类型转换用 astype 方法。例如：

```
int_arr = numpy.array([1,2,3,4,5])           #创建整型数组
float_arr = int_arr.astype(numpy.float64)      #转换为浮点 64 位数据
```



注意

astype 无论如何都会创建出一个新的数组，即原始数据的一份拷贝。

数据类型演示示例 5-2。

```
import numpy
print( '生成指定元素类型的数组:设置 dtype 属性')
x = numpy.array([1,2.6,3],dtype = numpy.int64)
print('x = numpy.array([1,2.6,3],dtype = numpy.int64)')
print( x,x.dtype)                      #元素类型为 int64
print('x = numpy.array([1,2,3],dtype = numpy.float64)')
x = numpy.array([1,2,3],dtype = numpy.float64)
print( x,x.dtype)                      #元素类型为 float64
print( '使用 astype 复制数组，并转换类型')
```



```

x = numpy.array([1,2.6,3],dtype = numpy.float64)
y = x.astype(numpy.int32)
print( y )                #[1 2 3]
print( x )                #[ 1.  2.6  3. ]
z = y.astype(numpy.float64)
print( z )                #[ 1.  2.  3.]
print( '将字符串元素转换为数值元素')
x = numpy.array(['1','2','3'],dtype = numpy.string_)
y = x.astype(numpy.int32)
print( x )                #[ '1' '2' '3' ]
print( y )                #[1 2 3] 若转换失败会抛出异常
print( '使用其他数组的数据类型作为参数')
x = numpy.array([ 1., 2.6,3. ],dtype = numpy.float32);
y = numpy.arange(3,dtype=numpy.int32);
print( y )                #[0 1 2]
print( y.astype(x.dtype)) #[ 0.  1.  2.]

```

程序运行结果:

生成指定元素类型的数组:设置 dtype 属性

```
x = numpy.array([1,2.6,3],dtype = numpy.int64)
```

```
[1 2 3] int64
```

```
x = numpy.array([1,2,3],dtype = numpy.float64)
```

```
[1. 2. 3.] float64
```

使用 astype 复制数组,并转换类型

```
[1 2 3]
```

```
[1. 2.6 3. ]
```

```
[1. 2. 3.]
```

将字符串元素转换为数值元素

```
[b'1' b'2' b'3']
```

```
[1 2 3]
```

使用其他数组的数据类型作为参数

```
[0 1 2]
```

```
[0. 1. 2.]
```

5.4 ndarray 数组的索引、切片和转置

1. ndarray 数组的基本索引和切片

ndarray 对象的内容可以通过索引或切片来访问和修改，与 Python 的内置容器对象一样。

多维数组每个轴有一个索引，这些索引由一个逗号分割的元组给出。例如：

```
arr[r1:r2, c1:c2]
arr[1,1] 等价 arr[1][1]
```

其中，[:]代表某个维度的数据。

下面见示例 5-3。

```
import numpy
print('ndarray 的基本索引')
x = numpy.array([[1,2],[3,4],[5,6]])
print('x = numpy.array([[1,2],[3,4],[5,6]])')
print('x[0]= ', x[0])                # [1,2]
print('x[0][1]= ', x[0][1])          # 2, 普通 Python 数组的索引
print('x[0,1]= ', x[0,1])            # 同 x[0][1], ndarray 数组的索引
x = numpy.array([[1, 2], [3,4]], [[5, 6], [7,8]])
print('x = numpy.array([[1, 2], [3,4]], [[5, 6], [7,8]])')
print('x[0]= ', x[0])                # [[1 2], [3 4]]
y = x[0].copy()                      # 生成一个副本
print('y = x[0].copy()')              # 生成一个副本'
z = x[0]                             # 未生成一个副本
print('z = x[0]')                    # 未生成一个副本')
print('y= ', y)                      # [[1 2], [3 4]]
print('y[0,0]= ', y[0,0])            # 1
print('y[0,0] = 0 ; z[0,0] = -1')
y[0,0] = 0
z[0,0] = -1
print('y= ', y)                      # [[0 2], [3 4]]
print('x[0]= ', x[0])                # [[-1 2], [3 4]]
print('z= ', z)                      # [[-1 2], [3 4]]
```



```

print( 'ndarray 的切片')
print('x = numpy.array([1,2,3,4,5])')
x = numpy.array([1,2,3,4,5])
print('x[1:3]= ', x[1:3])           # [2,3] 右边开区间
print('x[:3]= ', x[:3])             # [1,2,3] 左边默认为 0
print('x[1:]= ', x[1:])             # [2,3,4,5] 右边默认为元素个数
print('x[0:4:2]= ', x[0:4:2])       # [1,3] 下标递增 2
x = numpy.array([[1,2],[3,4],[5,6]])
print('x = numpy.array([[1,2],[3,4],[5,6]])')
print('x[:2]= ', x[:2])             # [[1 2],[3 4]]
print('x[:2,:1]= ', x[:2,:1])       # [[1],[3]]
x[:2,:1] = 0                        # 用标量赋值
print('x[:2,:1] = 0')               # 用标量赋值')
print( 'x= ',x)                     # [[0,2],[0,4],[5,6]]
x[:2,:1] = [[8],[6]]               # 用数组赋值
print('x[:2,:1] = [[8],[6]]')       # 用数组赋值')
print('x=', x)                     # [[8,2],[6,4],[5,6]]

```

程序运行结果:

```

ndarray 的基本索引
x = numpy.array([[1,2],[3,4],[5,6]])
x[0]= [1 2]
x[0][1]= 2
x[0,1]= 2
x = numpy.array([[[1, 2], [3,4]], [[5, 6], [7,8]]])
x[0]= [[1 2]
[3 4]]
y = x[0].copy()                    # 生成一个副本
z = x[0]                          # 未生成一个副本
y= [[1 2]
[3 4]]
y[0,0]= 1
y[0,0] = 0 ;z[0,0] = -1
y= [[0 2]
[3 4]]
x[0]= [[-1 2]
[ 3 4]]

```

```

z=  [[-1  2]
      [ 3  4]]
ndarray 的切片
x = numpy.array([1,2,3,4,5])
x[1:3]=  [2 3]
x[:3]=  [1 2 3]
x[1:]=  [2 3 4 5]
x[0:4:2]=  [1 3]
x = numpy.array([[1,2],[3,4],[5,6]])
x[:2]=  [[1 2]
          [3 4]]
x[:2,:1]=  [[1]
             [3]]
x[:2,:1] = 0          #用标量赋值
x=  [[0 2]
      [0 4]
      [5 6]]
x[:2,:1] =  [[8],[6]]    #用数组赋值
x=  [[8 2]
      [6 4]
      [5 6]]

```

2. ndarray 数组的布尔索引和花式索引

ndarray 数组除了前面介绍的普通索引和切片外，还支持布尔索引和花式索引。

(1) 布尔索引：使用布尔数组作为索引，如 `arr[condition]`，其中 `condition` 为由一个条件或多个条件组成的布尔数组。

布尔索引代码见示例 5-4。

```

import numpy
print( 'ndarray 的布尔索引')
x = numpy.array([3,2,3,1,3,0])
print('x = numpy.array([3,2,3,1,3,0])')
#布尔数组的长度必须跟被索引的轴长度一致
y = numpy.array([True,False,True,False,True,False])
print('y = numpy.array([True,False,True,False,True,False]) ')
print('x[y]= ', x[y]) #[3,3,3]

```

```

print('x[y==False]= ', x[y==False])#[2,1,0]
print('x>=3 ', x>=3)                #[ True False  True False  True  False]
print('x[~(x>=3)]= ', x[~(x>=3)])  #[2,1,0]
print(' (x==2)|(x==1)',(x==2)|(x==1))#[False True False  True False False]
print('x[(x==2)|(x==1)] ', x[(x==2)|(x==1)]) #[2 1]
x[(x==2)|(x==1)] = 0
print('x[(x==2)|(x==1)] = 0')
print('x= ', x) #[3 0 3 0 3 0]

```

程序运行结果:

```

ndarray 的布尔索引
x = numpy.array([3,2,3,1,3,0])
y = numpy.array([True,False,True,False,True,False])
x[y]=  [3 3 3]
x[y==False]=  [2 1 0]
x>=3  [ True False  True False  True False]
x[~(x>=3)]=  [2 1 0]
(x==2)|(x==1) [False  True False  True False False]
x[(x==2)|(x==1)]  [2 1]
x[(x==2)|(x==1)] = 0
x=  [3 0 3 0 3 0]

```

(2) 花式索引: 使用整型数组作为索引。

花式索引代码见示例 5-5。

```

import numpy
print(' ndarray 的花式索引:使用整型数组作为索引')
x = numpy.array([1,2,3,4,5,6])
print('x = numpy.array([1,2,3,4,5,6])')
print('x[[0,1,2]]= ', x[[0,1,2]]) #[1 2 3]
print('print x[[-1,-2,-3]]= ', x[[-1,-2,-3]]) #[6,5,4]
x = numpy.array([[1,2],[3,4],[5,6]])
print('x = numpy.array([[1,2],[3,4],[5,6]])')
print('x[[0,1]]= ', x[[0,1]]) #[[1,2],[3,4]]
print('x[[0,1],[0,1]]= ', x[[0,1],[0,1]]) #[1,4] 打印 x[0][0]和 x[1][1]
print('x[[0,1][:],[0,1]]= ', x[[0,1][:],[0,1]]) #打印 01 行的 01 列
[[1,2],[3,4]]
#使用 numpy.ix_() 增强可读性

```

```
print('使用 numpy.ix_() 增强可读性')
print('x[numpy.ix_([0,1],[0,1])]= ', x[numpy.ix_([0,1],[0,1])]) #同上
打印 01 行的 01 列 [[1,2],[3,4]]
x[[0,1],[0,1]] = [0,0]
print('x[[0,1],[0,1]] = [0,0]')
print('x= ', x) #[[0,2],[3,0],[5,6]]
```

程序运行结果:

```
ndarray 的花式索引:使用整型数组作为索引
x = numpy.array([1,2,3,4,5,6])
x[[0,1,2]]= [1 2 3]
print x[[-1,-2,-3]]= [6 5 4]
x = numpy.array([[1,2],[3,4],[5,6]])
x[[0,1]]= [[1 2]
[3 4]]
x[[0,1],[0,1]]= [1 4]
x[[0,1]][:,[0,1]]= [[1 2]
[3 4]]
使用 numpy.ix_() 增强可读性
x[numpy.ix_([0,1],[0,1])]= [[1 2]
[3 4]]
x[[0,1],[0,1]] = [0,0]
x= [[0 2]
[3 0]
[5 6]]
```

3. ndarray 数组的转置和轴对换

数组的转置和轴对换只会返回源数据的一个视图，不会对源数据进行修改。
代码见示例 5-6。

```
import numpy
print( 'ndarray 数组的转置和轴对换')
k = numpy.arange(9) #[0,1,...,8]
m = k.reshape((3,3)) #改变数组的 shape 复制生成二维的，每个维度长度为 3 的数组
print('k = numpy.arange(9)')
print('m = k.reshape((3,3))')
print('k= ', k)      #[0 1 2 3 4 5 6 7 8]
```

```

print('m= ', m)                #[[0 1 2] [3 4 5] [6 7 8]]
#转置(矩阵)数组: T 属性 : mT[x][y] = m[y][x]
print('转置(矩阵)数组: T 属性 : mT[x][y] = m[y][x]')
print('m.T= ', m.T)            #[[0 3 6] [1 4 7] [2 5 8]]
#计算矩阵的内积 xTx
print('numpy.dot(m,m.T)= ', numpy.dot(m,m.T)) #numpy.dot 点乘
#高维数组的轴对象
k = numpy.arange(8).reshape(2,2,2)
print('k = numpy.arange(8).reshape(2,2,2)')
print('k= ', k)                #[[[0 1],[2 3]],[[4 5],[6 7]]]
print('k[1][0][0]= ', k[1][0][0])
#轴对换 transpose 参数:由轴编号组成的元组
m = k.transpose((1,0,2))      #m[y][x][z] = k[x][y][z]
print('m = k.transpose((1,0,2))')
print('m= ', m)                #[[[0 1],[4 5]],[[2 3],[6 7]]]
print('m[0][1][0]= ', m[0][1][0])
#轴对换 swapaxes (axes: 轴), 参数:一对轴编号
m = k.swapaxes(0,1) #将第一个轴和第二个轴对换 m[y][x][z] = k[x][y][z]
print('m = k.swapaxes(0,1)')
print('m= ', m)                #[[[0 1],[4 5]],[[2 3],[6 7]]]
print('m[0][1][0]= ', m[0][1][0])
#使用轴对换进行数组矩阵转置
m = numpy.arange(9).reshape((3,3))
print('m = numpy.arange(9).reshape((3,3))')
print('m= ', m)                #[[0 1 2] [3 4 5] [6 7 8]]
print('m.swapaxes(1,0)= ', m.swapaxes(1,0)) #[[0 3 6] [1 4 7] [2 5 8]]

```

程序运行结果:

```

ndarray 数组的转置和轴对换
k = numpy.arange(9)
m = k.reshape((3,3))
k=  [0 1 2 3 4 5 6 7 8]
m=  [[0 1 2]
     [3 4 5]
     [6 7 8]]
转置(矩阵)数组: T 属性 : mT[x][y] = m[y][x]
m.T=  [[0 3 6]

```

```
[1 4 7]
[2 5 8]]
numpy.dot(m,m.T)= [[ 5 14 23]
[14 50 86]
[23 86 149]]
k = numpy.arange(8).reshape(2,2,2)
k= [[[0 1]
[2 3]]

[[4 5]
[6 7]]]
k[1][0][0]= 4
m = k.transpose((1,0,2))
m= [[[0 1]
[4 5]]

[[2 3]
[6 7]]]
m[0][1][0]= 4
m = k.swapaxes(0,1)
m= [[[0 1]
[4 5]]

[[2 3]
[6 7]]]
m[0][1][0]= 4
m = numpy.arange(9).reshape((3,3))
m= [[0 1 2]
[3 4 5]
[6 7 8]]
m.swapaxes(1,0)= [[0 3 6]
[1 4 7]
[2 5 8]]
```

5.5 NumPy通用函数

ndarray 数组除了支持运算符 “+” “-” “*” “/” “%” 等外，还支持一些 NumPy

的通用函数 ufunc。

NumPy 提供常见的数学函数如 `sin`, `cos` 和 `exp`, 这些被称作通用函数 ufunc。在 NumPy 中, 这些函数按数组的元素运算产生一个数组作为输出。

通用函数 ufunc 包括如下函数。

一元通用函数: `abs`, `fabs`, `sqrt`, `square`, `exp`, `log`, `log10`, `log2`, `log1p`, `sign`, `ceil`, `floor`, `rint`, `modf`, `isnan`, `isfinite`, `isinf`, `cos`, `cosh`, `sin`, `sinh`, `tan`, `tanh`, `arccos`, `arccosh`, `arcsin`, `arcsinh`, `arctan`, `arctanh`, `logical_not` 等。

二元通用函数: `add`, `subtract`, `multiply`, `divide`, `floor_divide`, `power`, `maximum`, `fmax`, `minimum`, `fmin`, `mod`, `copysign`, `greater`, `greater_equal`, `less`, `less_equal`, `equal`, `not_equal`, `logical_and`, `logical_or`, `logical_xor` 等。

选择或统计通用函数: `where`, `sum`, `mean`, `std`, `var`, `min`, `max`, `argmin`, `argmax`, `cumsum`, `cunprod`, `any`, `all` 等。

去重及集合运算函数: `unique(x)`, `intersect1d(x,y)`, `union1d(x,y)`, `in1d(x,y)`, `setdiff1d(x,y)`, `setxor1d(x,y)` 等。

以上函数的用法可在 NumPy 官网查询或者查看帮助 `help(numpy)`。

见示例 5-7。

```
import numpy
print( '一元 ufunc 示例' )
x = numpy.arange(6)
print('x = numpy.arange(6)')
print('x= ', x) #[0 1 2 3 4 5]
print('numpy.square(x)= ', numpy.square(x)) #[ 0  1  4  9 16 25]
print( '二元 ufunc 示例' )
x = numpy.array([[1,4],[6,7],[8,9]])
y = numpy.array([[2,3],[5,8],[10,11]])
print('x = numpy.array([[1,4],[6,7],[8,9]])')
print('y = numpy.array([[2,3],[5,8],[10,11]])')
print('numpy.maximum(x,y)= ', numpy.maximum(x,y))
print('numpy.minimum(x,y)= ', numpy.minimum(x,y))
print('对每一行的元素求平均 x.mean(axis=1)= ', x.mean(axis=1))
print('对每一列的元素求平均 x.mean(axis=0)= ', x.mean(axis=0))
```

程序运行结果:

```
一元 ufunc 示例
x = numpy.arange(6)
x= [0 1 2 3 4 5]
numpy.square(x)= [ 0  1  4  9 16 25]
二元 ufunc 示例
x = numpy.array([[1,4],[6,7],[8,9]])
y = numpy.array([[2,3],[5,8],[10,11]])
numpy.maximum(x,y)= [[ 2  4]
[ 6  8]
[10 11]]
numpy.minimum(x,y)= [[1  3]
[5  7]
[8  9]]
对每一行的元素求平均 x.mean(axis=1)= [2.5 6.5 8.5]
对每一列的元素求平均 x.mean(axis=0)= [5.          6.66666667]
```

5.6 ndarray数组文件的保存和读取

NumPy 提供了多种存取数组内容的文件操作函数。保存数组数据的文件可以是二进制格式也可以是文本格式,其中二进制格式的文件又分为 NumPy 专用的格式化类型和无格式类型。下面我们介绍这些方法的使用。

1. 二进制格式文件

把文件保存为二进制格式:

```
numpy.save(filename,ndarray)
```

读取二进制格式的文件:

```
ndarray=numpy.load(filename)
```

其中, `filename` 为文件名, `.npy` 为默认扩展名, `ndarray` 为数组名。在 `numpy.load(filename)` 中必须写上文件扩展名 `.npy`, 否则会报错。

如果将多个数组保存到一个文件中, 则可以使用 `numpy.savez()` 函数。

格式:

```
numpy.savez(filename, arr1, arr2, ...)
```

其中, filename 为文件名, .npz 为默认扩展名, arr1 和 arr2 为数组名。

numpy.load()函数可以自动识别.npz 文件, 并且返回一个类似于字典的对象, 也可以通过数组名获取数组的内容。例如:

```
r=numpy.load('aa.npz')
```

2. 文本格式文件

文本格式文件保存命令:

```
numpy.savetxt(fname, X, fmt='%18e', delimiter=' ', newline='\n',
header='', footer='', comments='#')
```

文本格式文件读取命令:

```
ndarray=numpy.loadtxt(fname, dtype=<class 'float'>, comments='#',
delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False,
ndmin=0)
```

3. tofile()函数和 fromfile()函数

将数组中的数据以二进制的格式存入文件, 命令如下:

```
ndarray.tofile(filename)
```

将数组中的数据以二进制的格式存入文件, 但输出的数组并不保存数组形状和元素类型等信息。文件名后缀一般选.bin。

读入数据并保存到数组中, 命令如下:

```
ndarray=numpy.fromfile(filename, dtype)
```

当 numpy.fromfile()函数读入数据时, 需要用户明确指定元素类型。

有关文件的保存和读取见示例 5-8。

```
import numpy
arr = numpy.arange(10)
print('arr = numpy.arange(10)')
print('save 和 load 演示')
```

```
print('numpy.save(\'test\',arr)')
numpy.save('test',arr)
arr2=numpy.load('test.npy')
print('arr2=numpy.load(\'test.npy\')')
print('arr2= ',arr2)
print('\nsavetxt 和 loadtxt 演示')
print('numpy.savetxt(\'test2.txt\',arr,fmt=\'%10.5f\',delimiter=',\')')
numpy.savetxt('test2.txt',arr,fmt='%10.5f',delimiter=',')
arr3=numpy.loadtxt('test2.txt',delimiter=',')
print('arr3=numpy.loadtxt(\'test2.txt\',delimiter=',\')')
print('arr3= ',arr3)
print('\ntofile 和 fromfile 演示')
print('arr.tofile(\'test.bin\')')
arr.tofile('test.bin')
arr4=numpy.fromfile('test.bin',dtype=numpy.int32)
print('arr4=numpy.fromfile(\'test.bin\',dtype=numpy.int32) ')
print('arr4= ',arr4)
```

程序运行结果:

```
arr = numpy.arange(10)
save 和 load 演示
numpy.save('test',arr)
arr2=numpy.load('test.npy')
arr2=  [0  1  2  3  4  5  6  7  8  9]

savetxt 和 loadtxt 演示
numpy.savetxt('test2.txt',arr,fmt='%10.5f',delimiter=',')
arr3=numpy.loadtxt('test2.txt',delimiter=',')
arr3=  [0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]

tofile 和 fromfile 演示
arr.tofile('test.bin')
arr4=numpy.fromfile('test.bin',dtype=numpy.int32)
arr4=  [0  1  2  3  4  5  6  7  8  9]
```



6

第 6 章

Pandas 库与数据处理

Pandas 是 Python Data Analysis Library 的简称，是基于 NumPy 的一种工具，该工具是为了解决数据分析问题创建的。Pandas 纳入了大量的库和标准的数据模型，提供了能高效操作大型数据集所需的工具。本书使用的是 Pandas 0.23.4 版本。

6.1 Pandas 安装和使用

在 Linux 和 macOS 系统下的安装命令如下：

```
sudo pip install pandas
```

在 Windows 系统下的安装命令如下：

```
pip install pandas
```

Python 中引入 Pandas 库使用如下命令：

```
import pandas
```

使用别名“pd”引入 Pandas 库，使用如下命令：

```
import pandas as pd
```

在 Python 3 中使用 Pandas 库使用小写名称 pandas。



6.2 Pandas数据结构

Pandas 最核心的数据结构就是 Series 和 DataFrame。

Series: 一维数组，与 NumPy 中的一维数组类似，与 Python 基本的数据结构列表也很相近，其区别是列表中的元素可以是不同的数据类型，而 Series 中则只允许存储相同的数据类型，这样可以更有效地使用内存，提高运算效率。

Time-Series: 以时间为索引的 Series。

DataFrame: 二维的表格型数据结构，也称数据框。它的很多功能与 R 语言中的 data.frame 类似，也可以将 DataFrame 理解为 Series 的容器。

6.2.1 Series 的创建

Series 类似于一维数组与字典数据结构的结合。它由一组数据和数据相对应的数据索引标签组成，而这组数据和索引标签的基础都是一个一维 ndarray 数组，可将 index 索引理解为行索引。Series 的创建主要有以下三种方式：

1) 通过一维数组创建

由于 Series 是一维结构的数组，因此可以直接通过列表来创建。见示例 6-1：

```
import pandas as pd
series1 = pd.Series([1, 2, 3, 4,5],index=None)
print(series1)
```

输出结果如下：

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

说明：

- 输出的最后一行是 Series 的数据类型，这里的数据类型都是 int64。

- 数据在第二列输出，第一列是数据的索引，在 Pandas 中称为 Index。
- index 参数是可以省略的。如果不带 index 参数或者像例子中出现 “index=None”，Pandas 就会自动默认用 index 进行索引，类似于数组。

2) 通过字典的方式创建

通过字典的方式创建 Series，见示例 6-2。

```
import pandas as pd
series2 = pd.Series({'a': [1, 2, 3, 4], 'b': [5, 6, 7, 8], 'c': [9, 10, 11, 12]})
print(series2)
```

输出结果如下：

```
a      [1, 2, 3, 4]
b      [5, 6, 7, 8]
c      [9, 10, 11, 12]
dtype: object
```

6.2.2 DataFrame 的创建

DataFrame 是一个类似于表格的数据结构，包含一组有序的列，每列可以是不同的类型（数值、字符串、布尔值等）。它的索引包括列索引和行索引。DataFrame 的每一行和每一列都是一个 Series，Series 的 name 属性为当前数据框的行索引名或列索引名。

DataFrame 的创建主要有以下两种方式：

1) 通过二维数组创建

通过二维数组创建 DataFrame，见示例 6-3。

```
import pandas as pd
import numpy as np
df1 = pd.DataFrame(np.array(np.arange(12)).reshape(4, 3))
print(df1)
```

输出结果如下：

```
0  1  2
```

```
0 0 1 2
1 3 4 5
2 6 7 8
3 9 10 11
```

2) 通过字典的方式创建

通过字典方式创建 DataFrame，见示例 6-4。

```
import pandas as pd
df2 = pd.DataFrame({'a': [1, 2, 3, 4], 'b': [5, 6, 7, 8], 'c': [9, 10, 11, 12]})
print(df2)
```

输出结果如下：

```
   a  b  c
0  1  5  9
1  2  6 10
2  3  7 11
3  4  8 12
```

6.3 股票数据使用

我们可以通过网上一些金融数据 API 接口来获取，也可以自己写网络爬虫程序抓取网页中的股票行情数据。下面具体介绍通过金融数据 API 接口来获取数据。

目前，提供股票数据源的有聚宽 JQData 数据、QUANTAXIS 数据、Tushare 数据、OpenDataTools 数据等。

1. 聚宽 JQData 数据

首先要注册聚宽用户，记下用户名和密码，并申请 JQData 本地量化金融数据，然后安装 jqdatasdk 模块。具体操作如下：

1) 导入 JQData

```
from jqdatasdk import *
```

2) 用户认证

```
jqdatasdk.auth("用户名","密码")
```

在认证成功后, 如果显示“auth success”, 就可以使用数据了; 如果没有显示, 就要先检查一下是否已经申请并通过。

3) 获取数据

```
get_price(security, start_date=None, end_date=None, frequency='daily',
fields=None, skip_paused=False, fq='pre', count=None)
```

其中涉及的参数及说明见表 6-1。

表 6-1 get_price()函数的参数说明

| 参数 | 说明 |
|-------------|---|
| security | 一只股票代码或者一只股票代码的 list |
| count | 数量, 返回的结果集的行数, 即表示获取 end_date 之前有几个 frequency 的数据。与 start_date 不可同时使用 |
| start_date | 开始时间, 字符串或者[datetime.datetime]/[datetime.date]对象。如果参数 count 和 start_date 都没有, 则 start_date 生效, 默认值是'2015-01-01'。与 count 不可同时使用 |
| end_date | 结束时间, 格式同上, 默认是'2015-12-31', 包含此日期。注意: 当取分钟数据时, 如果 end_date 只有日期, 则日内时间等同于 00:00:00, 所以返回的数据是不包括 end_date 这一天的 |
| frequency | 单位时间长度, 几天或者几分钟, 现在支持'Xd', 'Xm', 'daily'等同于'1d', 'minute'等同于'1m', 其中 X 是一个正整数, 分别表示 X 天和 X 分钟 (不论是按天还是按分钟回溯都能得到这两种单位的数据)。注意: 当 X > 1 时, fields 只支持'open', 'close', 'high', 'low', 'volume', 'money' 标准字段, 默认值是 daily |
| fields | 字符串 list, 选择要获取的行情数据字段, 默认是 None (表示'open', 'close', 'high', 'low', 'volume', 'money'标准字段), 支持 SecurityUnitData 的所有基本属性, 包含'open', 'close', 'low', 'high', 'volume', 'money', 'factor', 'high_limit', 'low_limit', 'avg', 'pre_close', 'paused' |
| skip_paused | 是否跳过不交易的日期 (包括停牌、未上市或者退市后)。如果不跳过, 在停牌时就会使用停牌前的数据填充 (具体请看 SecurityUnitData 的 paused 属性), 而上市前或者退市后的数据都为 nan |
| fq | 'pre': 前复权; None: 不复权, 返回实际价格; 'post': 后复权 |

见示例 6-5。

```
import jqdatasdk as jq
x=jq.auth('18578755056', '18578755056')
data=jq.get_price('000001.XSHE',start_date='2015-01-01',end_date='201
```

```
8-10-16')
print(data.tail(10)) #输出最后十条数据
```

程序运行结果:

| | open | close | high | low | volume | money |
|------------|-------|-------|-------|-------|-------------|--------------|
| 2018-09-26 | 10.61 | 10.71 | 10.92 | 10.52 | 149051343.0 | 1.600626e+09 |
| 2018-09-27 | 10.65 | 10.74 | 10.84 | 10.57 | 88036333.0 | 9.422004e+08 |
| 2018-09-28 | 10.78 | 11.05 | 11.27 | 10.78 | 211024267.0 | 2.331358e+09 |
| 2018-10-08 | 10.70 | 10.45 | 10.79 | 10.45 | 168635852.0 | 1.793455e+09 |
| 2018-10-09 | 10.46 | 10.56 | 10.70 | 10.39 | 106408426.0 | 1.117947e+09 |
| 2018-10-10 | 10.54 | 10.45 | 10.66 | 10.38 | 99520008.0 | 1.045666e+09 |
| 2018-10-11 | 10.05 | 9.86 | 10.16 | 9.70 | 199514383.0 | 1.994187e+09 |
| 2018-10-12 | 9.97 | 10.30 | 10.34 | 9.87 | 151681028.0 | 1.532651e+09 |
| 2018-10-15 | 10.39 | 10.11 | 10.47 | 10.09 | 140268530.0 | 1.443043e+09 |
| 2018-10-16 | 10.14 | 10.37 | 10.54 | 10.12 | 139692105.0 | 1.450020e+09 |

2. QUANTAXIS 数据

QUANTAXIS 数据获取日线数据的方法:

```
data= QUANTAXIS.QA_fetch_get_stock_day(tdx',code,start_date,
end_date,'00')
```

参数都是字符串型,其中 code 是股票代码、start_date 是起始日期、end_date 是结束日期、第四个参数选择复权方式。

见示例 6-6。

```
import QUANTAXIS as QA #导入模块
df2=QA.QA_fetch_get_stock_day('tdx','600020','2018-08-01','2018-12-03',
'00')#获取股票数据
print('\ndf2.head()\n',df2.head())
```

程序运行结果:

```
df2.head()
open close high      ...              date      code      date_stamp
date                               ...
2018-08-01  3.81   3.76  3.84      ...  2018-08-01  600020  1.533053e+09
2018-08-02  3.76   3.66  3.76      ...  2018-08-02  600020  1.533139e+09
```



```
2018-08-03  3.66   3.66  3.70   ...  2018-08-03  600020  1.533226e+09
2018-08-06  3.66   3.62  3.69   ...  2018-08-06  600020  1.533485e+09
2018-08-07  3.61   3.73  3.75   ...  2018-08-07  600020  1.533571e+09
```

```
[5 rows x 9 columns]
```

```
索引类型 type(df2.index)  <class 'pandas.core.indexes.base.Index'>
```

3. Tushare 数据

Tushare 数据源模块的安装:

```
pip install tushare
```

获取行情数据的方法如下:

```
import tushare as ts
ts.get_k_data(code=None, start='', end='', ktype='D', autype='qfq',
index=False, retry_count=3, pause=0.001)
```

Tushare 获取 K 线数据的参数及说明见表 6-2。

表 6-2 get_k_data()函数的参数说明

| 参数 | 说明 |
|-------------|---|
| code | string, 股票代码如 600848 |
| start | string, 当开始日期(格式: YYYY-MM-DD)为空时取当前日期 |
| end | string, 当结束日期(格式: YYYY-MM-DD)为空时取去年今日 |
| autype | string, 复权类型: qfq 为前复权、hfq 为后复权、None 为不复权, 默认为 qfq |
| ktype | string, 数据类型: D=日 K 线、W=周、M=月、5=5 分钟、15=15 分钟、30=30 分钟、60=60 分钟, 默认为 D |
| index | bool, 默认为 False, 表示不是指数; 值为 True, 表示是指数 |
| retry_count | int, 默认为 3, 如遇网络等问题重复执行的次数 |
| pause | int, 默认为 0, 在重复请求数据过程中暂停的秒数, 防止请求间隔时间太短出现问题 |
| drop_factor | bool, 默认为 True, 是否移除复权因子。在分析过程中, 可能复权因子意义不大, 但是如果先储存到数据库再去分析, 该项目就会更加灵活 |

get_k_data()函数的使用见示例 6-7。

```
import tushare as ts
data = ts.get_k_data('600099', ktype='D')
print(data.tail(10))          #输出最后十条数据
```

程序运行结果:

| | date | open | close | high | low | volume | code |
|-----|------------|------|-------|------|------|---------|--------|
| 631 | 2019-06-11 | 7.35 | 7.44 | 7.46 | 7.18 | 51453.0 | 600099 |
| 632 | 2019-06-12 | 7.44 | 7.50 | 7.66 | 7.38 | 57263.0 | 600099 |
| 633 | 2019-06-13 | 7.50 | 7.69 | 7.92 | 7.44 | 66571.0 | 600099 |
| 634 | 2019-06-14 | 7.60 | 7.34 | 7.67 | 7.33 | 41663.0 | 600099 |
| 635 | 2019-06-17 | 7.33 | 7.54 | 7.54 | 7.29 | 40202.0 | 600099 |
| 636 | 2019-06-18 | 7.55 | 7.43 | 7.60 | 7.37 | 27838.0 | 600099 |
| 637 | 2019-06-19 | 7.50 | 7.65 | 7.70 | 7.50 | 50161.0 | 600099 |
| 638 | 2019-06-20 | 7.65 | 7.75 | 7.81 | 7.49 | 58359.0 | 600099 |
| 639 | 2019-06-21 | 7.90 | 7.93 | 7.96 | 7.78 | 75712.0 | 600099 |
| 640 | 2019-06-24 | 7.95 | 8.09 | 8.13 | 7.81 | 76023.0 | 600099 |

4. OpenDataTools 数据

OpenDataTools 数据的使用说明见官网。

见示例 6-8。

```
from opendatatools import stock
df, msg = stock.get_quote('600000.SH,000002.SZ')
print('索引类型 type(df.index) ', type(df.index))
print(df)
df1, msg = stock.get_daily('600000.SH',
                           start_date='2018-01-01',end_date='2018-08-17')
print(df1.tail())
```

程序运行结果:

```
索引类型 type(df.index) <class 'pandas.core.indexes.range.RangeIndex'>
      amount  change  ...  turnover_rate  volume
0  7.592698e+08  -0.18  ...           0.28  27012891
1  4.111724e+08  -0.06  ...           0.12  34195336

[2 rows x 13 columns]
      change  high  last  ...  time  turnover_rate  volume
16 -0.0869  9.7566  9.6790  ...  2018-08-13           0.06  17927668
17  0.0774  9.7663  9.7566  ...  2018-08-14           0.06  15868585
18 -0.1551  9.7566  9.6014  ...  2018-08-15           0.05  14787033
```

```

19  0.1555  9.8148  9.7566  ...  2018-08-16  0.09  24323652
20 -0.0781  9.8536  9.6790  ...  2018-08-17  0.05  15090455

[5 rows x 10 columns]

```

在 OpenDataTools 数据表格中用 last 代替了 close，使用前要先更换列名。

上面介绍了几种获取股票历史数据的方法，其实不同方法获取的股票数据都是一样的，区别就是 DataFrame 的索引和表头（也叫列名）不同。后面我们将会学习怎样转换为统一格式，以方便数据的计算和图形、图表的显示。

6.4 DataFrame 数据操作

1. DataFrame 数据的基本提取

DataFrame 数据的基本操作主要涉及数据显示方面的操作，假定 DataFrame 数据名为 df1。

(1) 查看全部数据，可以用 print() 函数，例如：

```
print(df1)
```

(2) 查看 1 列的命令格式：

```
DataFrame.Column
```

查看名为 df1 的 close 列的命令：

```
print(df1['close '])
print(df1.close)
```

查看多个列的命令：

```
DataFrame[['column1 ', 'column2 ', ... 'columnN ']]
```

查看 df1 中多个列的命令：

```
print(df1[['date ', 'close ', 'high ']])
```

(3) 查看前几行的命令格式：

```
DataFrame.head(n=5)
```

参数 `n` 是开始多少行，默认为 5，如查看前 6 行：

```
print(df1.head(6))      #查看前 6 行
```

(4) 查看最后几行的命令格式：

```
DataFrame.tail(n=5)
```

参数 `n` 是最后多少行，默认为 5，如查看最后 2 行：

```
print(df1.tail(2))      #查看最后 2 行
```

(5) 查看所有数据值的命令格式：

```
DataFrame.values
```

返回值为 `numpy.ndarray` 类型。例如：

```
print(df1.values)
```

(6) 查看所有列名的命令格式：

```
DataFrame.columns
```

返回值为列名的列表，见示例 6-9。

```
import tushare as ts
df1 = ts.get_k_data('000776', ktype='D')
bb= df1. columns      #获取列名
for c in bb:          #循环打印列名
    print(c, end=' ')
```

程序运行结果：

```
date open close high low volume code
```

(6) 查看索引列的命令格式：

```
DataFrame.index
```

返回值为索引值的列表。例如：

```
print(df1. index)
```

(7) 行列转置的命令格式：

```
DataFrame.T
```

例如，df1 的行列转置命令如下。

```
print(df1.T)
```

(8) 修改列名的命令格式：

```
DataFrame.rename(mapper=None, index=None, columns=None, axis=None,
copy=True, inplace=False, level=None)
```

其中涉及的参数及含义如下：

mapper, index, columns: 映射的规则。

axis: 指定轴，可以是轴名称 `index` 或 `columns`，或数字 0 或 1，默认为 `index`。

copy: 布尔值，默认为 `True`，复制底层数据。

inplace: 布尔值，默认为 `False`，指定是否返回新的 `DataFrame`。如果为 `True`，则在原 `DataFrame` 数据上修改，返回值为 `None`。

level: `int` 或 `level name`，默认为 `None`。如果是 `MultiIndex`，就只重命名指定级别的标签。

更换列名，见示例 6-10。

```
import pandas as pd
df = pd.DataFrame({'a':[1,2,3], 'b':[1,2,3]})
print(df)
#1.修改列名 a, b 为 A, B。
df.columns = ['A','B']
print(df)
#2.只修改列名 a 为 A
df2=df.rename(columns={'A':'AA'})
print(df2)
```

2. DataFrame 数据转化为 Series 数据

`DataFrame` 数据为多列名，`Series` 数据为单列名，可以用下面命令将 `DataFrame` 数据转化为 `Series` 数据。

命令格式 1：

```
Series=DataFrame.column1
```

命令格式 2:

```
Series=DataFrame['column1']
```

见示例 6-11。

```
import tushare as ts
data = ts.get_k_data('600099',ktype='D')
print('import tushare as ts')
print('data = ts.get_k_data(\'600099\',ktype=\'D\')')
s1=data['close']
print('s1=data[\'close\'] ,type(s1)= ',type(s1))
print(s1.head())
s2=data.open
print('s2=data.open , type(s2)= ',type(s2))
print(s2.tail())
print('id(s1),id(s2) : ',id(s1),id(s2))
```

程序运行结果:

```
import tushare as ts
data = ts.get_k_data('600099',ktype='D')
s1=data['close'] ,type(s1)= <class 'pandas.core.series.Series'>
0    11.324
1    11.384
2    11.852
3    11.982
4    12.011
Name: close, dtype: float64
s2=data.open , type(s2)= <class 'pandas.core.series.Series'>
635    5.85
636    5.91
637    5.78
638    5.81
639    5.76
Name: open, dtype: float64
1    11.384
2    11.852
```

```
Name: close, dtype: float64
id(s1),id(s2) : 288982016 288982016
```

3. DataFrame 数据的复制

由于 DataFrame 和 Series 的数据类型都是对象数据，因此不能用简单的变量赋值方式进行复制，只能使用 `copy()` 方法操作。

命令格式：

```
DataFrame2=DataFrame.copy()
```

返回的 DataFrame2 是新创建的 DataFrame 数据。

假定 df1 是 DataFrame 数据。如果 df2=df1，则 id(df1)=id(df2)，即它们共用一份数据，因此不论是 df1 还是 df2 被修改，数据都会改变。假如要创建新副本数据，就要用 df3=df1.copy() 函数，此时 id(df3)!=id(df1)，它们分别是不同位置的数据。

4. DataFrame 数据的对列切片

DataFrame 数据可以进行切片操作或提取（截取）数列操作。

提取数列操作格式如下，注意要用双中括号 “[[]]”。

```
DataFrame2= DataFrame[['column1',' column2',... 'columnN']]
```

返回的 DataFrame2 是新创建的 DataFrame 数据。

提取单列命令格式如下：

```
DataFrame3= DataFrame[['column1']]
```

返回的 DataFrame3 是新创建的 DataFrame 数据。



注意

如果单列名用单中括号，返回的就是 Series 数据，如 `Series=DataFrame['column1']`。

DataFrame 数据后面可以任意增加新列，但是 Series 数据不可以，因为它的定义就是一维数组。

5. DataFrame 数据的对行切片

DataFrame 数据对行切片与列表的操作一样。

DataFrame 数据切片命令格式如下：

```
DataFrame2= DataFrame[index_start,index_end]
```

返回新的 DataFrame 数据。



注意

数据行中包含 `index >= index_start` 且 `index < index_end`，也就是说在数据中不包含 `index_end` 行。

例如，`print(d3[1:3])`的执行结果。

| | date | open | close | high | low | volume | code |
|---|------------|--------|--------|--------|--------|---------|--------|
| 1 | 2016-05-16 | 10.011 | 10.319 | 10.329 | 9.991 | 46384.0 | 600080 |
| 2 | 2016-05-17 | 10.229 | 10.070 | 10.339 | 10.041 | 42109.0 | 600080 |

6. DataFrame 数据的 loc[]和 iloc[]操作

DataFrame.loc[]通过标签来选择数据，其格式如下：

```
DataFrame2= DataFrame.loc[row_indexer,column_indexer]
```

提取一列格式：

```
DataFrame.loc[:, 'column']
```

提取多列格式：

```
DataFrame.loc[:, 'column1' : 'columnN']
```

提取特定行列格式：

```
DataFrame.loc[index1 : indexM , 'column1' : 'columnN']
```

例如，`df.loc[1:3,'date':'low']`会返回索引“1：3”之间的数据，以及列名“date”至列名“low”之间的数据，它们不是两列，而是包含在之间的很多列。

命令 `print(df.loc[1:3,'date':'low'])`的运行结果：

| | date | open | close | high | low |
|---|------------|--------|--------|--------|--------|
| 1 | 2016-05-16 | 10.011 | 10.319 | 10.329 | 9.991 |
| 2 | 2016-05-17 | 10.229 | 10.070 | 10.339 | 10.041 |
| 3 | 2016-05-18 | 10.031 | 9.673 | 10.031 | 9.464 |

`DataFrame.iloc[]`通过位置索引来选择数据，其格式如下：

```
DataFrame2= DataFrame.iloc[row_number,column_number]
```

这里的参数是数字位置范围。

例如，`df.iloc[:2,4]`的运行结果。

| | date | open | close | high |
|---|------------|--------|--------|--------|
| 0 | 2016-05-13 | 9.792 | 9.981 | 10.031 |
| 1 | 2016-05-16 | 10.011 | 10.319 | 10.329 |

7. DataFrame 数据的 at[]和 iat[]操作

`DataFrame` 数据的 `at[]`和 `iat[]`操作同 `loc[]`和 `iloc[]`操作，区别是 `loc[]`和 `iloc[]`获取的是一组数据，而 `at[]`和 `iat[]`获取的是一个数值。

旧版本中 `DataFrame` 行列数值获取使用的 `DataFrame.get_value(index,column)`方法都需要换为 `DataFrame.at[index,column]`方法。

`at` 格式：

```
DataFrame2= DataFrame.at[index,column]
```

`iat` 格式：

```
DataFrame2= DataFrame.iat[index,column_number]
```

例如，`print(df.at[3,'low'])`的运行结果。

```
9.464
print(df.iat[2,4])的运行结果。
10.329
```

8. DataFrame 数据的条件提取

我们可以按照一些条件来获取满足要求的数据。

条件提取数据的格式:

```
DataFrame2= DataFrame [condition]
```

其中, **condition** 是关于 **DataFrame** 数据的逻辑条件。

例如, 利用 `print(d1[d1.close>11].head())` 把 `close>11` 的数据提取出来。

| | date | open | close | high | low | volume | code |
|----|------------|--------|--------|--------|--------|----------|--------|
| 26 | 2016-06-22 | 10.836 | 11.472 | 11.641 | 10.766 | 130506.0 | 600080 |
| 27 | 2016-06-23 | 11.432 | 11.422 | 11.601 | 11.313 | 69843.0 | 600080 |
| 28 | 2016-06-24 | 11.452 | 11.273 | 11.512 | 10.866 | 78416.0 | 600080 |
| 29 | 2016-06-27 | 11.114 | 11.412 | 11.482 | 11.094 | 63153.0 | 600080 |
| 30 | 2016-06-28 | 11.184 | 11.532 | 11.591 | 11.184 | 66206.0 | 600080 |

完整程序见示例 6-12。

```
import tushare as ts
data = ts.get_k_data('600080',ktype='D')
print('import tushare as ts')
print('data = ts.get_k_data(\'600099\',ktype=\'D\')')
print(data.head(3))
d1=data
print('id(d1),id(data) : ',id(d1),id(data))
#截取单列 DataFrame 数据, 注意是加双中括号
d2=data[['close']]
print('type(d2) ',type(d2))
#截取多列 DataFrame 数据, 注意是加双中括号
d3=data[['date','close','open']]
print(d3.head(3))
print(d1[1:3])
print(d1.loc[1:3,'date':'low'])
print(d1.iloc[:2,:4])
print(d1[d1.close>11].head())
```

程序运行结果:

```
import tushare as ts
data = ts.get_k_data('600099',ktype='D')
date open close high low volume code
0 2016-05-13 9.792 9.981 10.031 9.792 32052.0 600080
```

```

1 2016-05-16 10.011 10.319 10.329 9.991 46384.0 600080
2 2016-05-17 10.229 10.070 10.339 10.041 42109.0 600080
id(d1),id(data) : 282547480 282547480
type(d2) <class 'pandas.core.frame.DataFrame'>
      date  close  open
0 2016-05-13  9.981  9.792
1 2016-05-16 10.319 10.011
2 2016-05-17 10.070 10.229
      date  open  close  high  low  volume  code
1 2016-05-16 10.011 10.319 10.329 9.991 46384.0 600080
2 2016-05-17 10.229 10.070 10.339 10.041 42109.0 600080
      date  open  close  high  low
1 2016-05-16 10.011 10.319 10.329 9.991
2 2016-05-17 10.229 10.070 10.339 10.041
3 2016-05-18 10.031  9.673 10.031  9.464
      date  open  close  high
0 2016-05-13  9.792  9.981 10.031
1 2016-05-16 10.011 10.319 10.329
      date  open  close  high  low  volume  code
26 2016-06-22 10.836 11.472 11.641 10.766 130506.0 600080
27 2016-06-23 11.432 11.422 11.601 11.313  69843.0 600080
28 2016-06-24 11.452 11.273 11.512 10.866  78416.0 600080
29 2016-06-27 11.114 11.412 11.482 11.094  63153.0 600080
30 2016-06-28 11.184 11.532 11.591 11.184  66206.0 600080

```

9. DataFrame 数据的排序

我们可以按索引值（或列名）和列值对 DataFrame 数据或列数据进行排序，生成新的数据。

（1）按照索引值（或列名）排序的格式：

```

DataFrame2= DataFrame.sort_index(axis=0, level=None, ascending=True,
inplace=False, kind='quicksort', na_position='last', sort_remaining=True,
by=None)

```

DataFrame.sort_index 参数的种类及说明见表 6-3。

表 6-3 DataFrame.sort_index 参数说明

| 参数 | 说明 |
|-------------|---|
| axis | 0 按照行名排序, 1 按照列名排序 |
| level | 默认 None, 否则按照给定的 level 顺序排列 |
| ascending | 默认 True 为升序排列, False 为降序排列 |
| inplace | 默认 False, 否则排序之后的数据直接替换原来的数据框 |
| kind | 默认 quicksort, 排序的方法 |
| na_position | 缺失值默认排在最后 {"first","last"} |
| by | 按照列数据进行排序, 单列用 by='column', 多列用 by= ['column1','column2'] |

有参数的排序, 例如:

```
print(d1.sort_index(axis=0,ascending=True,by=['close','low']))
```

无参数的是按索引来排序的, 例如:

```
print(d1.sort_index())
```

(2) 按列值排序的格式如下:

```
DataFrame2= DataFrame.sort_values(by, axis=0, ascending=True,
inplace=False, kind='quicksort', na_position='last')
```

DataFrame.sort_values 参数的种类及说明见表 6-4。

表 6-4 DataFrame.sort_values 参数说明

| 参数 | 说明 |
|-------------|---|
| Axis | 0 按照行名排序, 1 按照列名排序 |
| Level | 默认 None, 否则按照给定的 level 顺序排列 |
| ascending | 默认 True 为升序排列, False 为降序排列 |
| inplace | 默认 False, 否则排序之后的数据直接替换原来的数据框 |
| kind | 默认 quicksort, 排序的方法 |
| na_position | 缺失值默认排在最后 {"first","last"} |
| by | 按照列数据进行排序, 单列用 by='column', 多列用 by= ['column1','column2'] |

见示例 6-13。

```
import tushare as ts
d1 = ts.get_k_data('600080',ktype='D')
```

```
#print(d1.sort_index())
print(d1.sort_values(by=['close','low']).head(3))
print(d1.sort_values(['close','low']).head(3))
```

程序运行结果:

| | date | open | close | high | low | volume | code |
|-----|------------|------|-------|------|------|----------|--------|
| 639 | 2018-12-28 | 6.28 | 6.16 | 6.37 | 6.12 | 123266.0 | 600080 |
| 638 | 2018-12-27 | 6.50 | 6.23 | 6.65 | 6.23 | 173570.0 | 600080 |
| 637 | 2018-12-26 | 6.52 | 6.43 | 6.57 | 6.36 | 110473.0 | 600080 |

| | date | open | close | high | low | volume | code |
|-----|------------|------|-------|------|------|----------|--------|
| 639 | 2018-12-28 | 6.28 | 6.16 | 6.37 | 6.12 | 123266.0 | 600080 |
| 638 | 2018-12-27 | 6.50 | 6.23 | 6.65 | 6.23 | 173570.0 | 600080 |
| 637 | 2018-12-26 | 6.52 | 6.43 | 6.57 | 6.36 | 110473.0 | 600080 |

10. DataFrame 数据的插入

在指定位置插入一列数据的格式:

```
DataFrame.insert(loc, column, value, allow_duplicates=False)
```

其中, 参数 **column** 是新列名, **value** 是数值, **Series** 数据或列表。

例如:

```
d1.insert(1, 'aa', 2.0)
```

如果是在列名末尾插入新列, 也可以进行如下操作。

```
d1['bb']=3.0
```

其中, **bb** 是新列名。如果存在相同的“bb”, 就会修改掉旧“bb”列的数值。

11. DataFrame 数据的删除

(1) 删除一列数据用 **del** 语句, 格式如下:

```
del DataFrame['column']
```

例如, 删除上例增加的“bb”列用 **del d1['bb']**, 也可以使用下面命令格式:

```
DataFrame2=DataFrame.drop('column',axis=1)
```

返回删除掉的列数据, 旧数据没有变化。

```
d4=d1.drop('aa',axis=1)
```

(2) 删除一行数据使用如下命令格式：

```
DataFrame.drop(labels=None,axis=0, index=None, columns=None, inplace=False)
```

其中，`DataFrame.drop` 参数的种类及说明见表 6-5。

表 6-5 `DataFrame.drop` 参数说明

| 参数 | 说明 |
|---------|---|
| labels | 要删除的行列的名字，用列表给定 |
| axis | 默认为 0，指删除行，因此在删除 columns 时要指定 <code>axis=1</code> |
| index | 直接指定要删除的行 |
| columns | 直接指定要删除的列 |
| inplace | 如果为 <code>False</code> ，则默认该删除操作不改变原数据，而是返回一个执行删除操作后的新 <code>DataFrame</code> ； 如果为 <code>True</code> ，则会直接在原数据上进行删除操作，删除后无法返回 |

因此，删除行列有两种方式：

“`labels=None,axis=0`” 的组合和用 `index` 或 `columns` 直接指定要删除的行或列。

12. DataFrame 数据类型的转换

`DataFrame` 转换数据类型的格式如下：

```
DataFrame.astype(dtype[, copy, errors])
```

例如：

```
d4.code=d4.code.astype('int')
```

见示例 6-14。

```
import pandas as pd
import tushare as ts
print(pd.__version__)
data = ts.get_k_data('600080',ktype='D')
print('import tushare as ts')
print('data = ts.get_k_data(\'600099\',ktype=\'D\')')
print(data.head(3))
dl=data
```

```

print('id(d1),id(data) : ',id(d1),id(data))
#截取单列 DataFrame 数据, 注意是加双中括号
d2=data[['close']]
print('type(d2) ',type(d2.head()))
#截取多列 DataFrame 数据, 注意是加双中括号
d3=data[['date','close','open']]
print(d3.head(3))
print(d1[1:3])
print(d1.loc[1:3,'date':'low'])
print(d1.iloc[:2,:4])
print(d1[d1.close>11].head())
#print(d1.sort_index(axis=0,ascending=True,by=['close','low']))
print(d1.sort_index().head())
print(d1.sort_values(by=['close','low']).head(3))
print(d1.sort_values(['close','low']).head(3))
d1.insert(1,'aa',2.0)
d1['bb']=3.0
d1['aa']=1.5
del d1['bb']
d4=d1.drop('aa',axis=1)
d4=d1.drop(2)
print(d4.head())
print('type(d4.code[1])= ',type(d4.code[1]))
d4.code=d4.code.astype('int')
print('type(d4.code[1])= ',type(d4.code[1]))

```

程序运行结果:

```

0.23.4
import tushare as ts
data = ts.get_k_data('600099',ktype='D')
      date    open  close   high    low  volume   code
0  2016-05-13   9.792   9.981  10.031   9.792  32052.0  600080
1  2016-05-16  10.011  10.319  10.329   9.991  46384.0  600080
2  2016-05-17  10.229  10.070  10.339  10.041  42109.0  600080
id(d1),id(data) :  291675888 291675888
type(d2)  <class 'pandas.core.frame.DataFrame'>
      date    close    open

```

```

0 2016-05-13  9.981  9.792
1 2016-05-16 10.319 10.011
2 2016-05-17 10.070 10.229
      date  open  close  high  low  volume  code
1 2016-05-16 10.011 10.319 10.329  9.991 46384.0 600080
2 2016-05-17 10.229 10.070 10.339 10.041 42109.0 600080
      date  open  close  high  low
1 2016-05-16 10.011 10.319 10.329  9.991
2 2016-05-17 10.229 10.070 10.339 10.041
3 2016-05-18 10.031  9.673 10.031  9.464
      date  open  close  high
0 2016-05-13  9.792  9.981 10.031
1 2016-05-16 10.011 10.319 10.329
      date  open  close  high  low  volume  code
26 2016-06-22 10.836 11.472 11.641 10.766 130506.0 600080
27 2016-06-23 11.432 11.422 11.601 11.313  69843.0 600080
28 2016-06-24 11.452 11.273 11.512 10.866  78416.0 600080
29 2016-06-27 11.114 11.412 11.482 11.094  63153.0 600080
30 2016-06-28 11.184 11.532 11.591 11.184  66206.0 600080
      date  open  close  high  low  volume  code
0 2016-05-13  9.792  9.981 10.031  9.792 32052.0 600080
1 2016-05-16 10.011 10.319 10.329  9.991 46384.0 600080
2 2016-05-17 10.229 10.070 10.339 10.041 42109.0 600080
3 2016-05-18 10.031  9.673 10.031  9.464 48166.0 600080
4 2016-05-19  9.673  9.732  9.881  9.673 33785.0 600080
      date  open  close  high  low  volume  code
639 2018-12-28  6.28  6.16  6.37  6.12 123266.0 600080
638 2018-12-27  6.50  6.23  6.65  6.23 173570.0 600080
637 2018-12-26  6.52  6.43  6.57  6.36 110473.0 600080
      date  open  close  high  low  volume  code
639 2018-12-28  6.28  6.16  6.37  6.12 123266.0 600080
638 2018-12-27  6.50  6.23  6.65  6.23 173570.0 600080
637 2018-12-26  6.52  6.43  6.57  6.36 110473.0 600080

```



```

      date    aa  open  close  high  low  volume  code
0 2016-05-13  1.5  9.792  9.981 10.031 9.792 32052.0 600080
1 2016-05-16  1.5 10.011 10.319 10.329 9.991 46384.0 600080
3 2016-05-18  1.5 10.031  9.673 10.031 9.464 48166.0 600080
4 2016-05-19  1.5  9.673  9.732  9.881 9.673 33785.0 600080
5 2016-05-20  1.5  9.722  9.842  9.852 9.543 28619.0 600080
type(d4.code[1])= <class 'str'>
type(d4.code[1])= <class 'numpy.int32'>

```

6.5 DataFrame无效值

在 DataFrame 数据中, 经常存在无效值, 显示为 NaN(Not a Number)。Pandas 提供了一些方法可以判断、操作这些无效值。

1. 无效值的判断

判断数据的缺失或无效值的命令如下:

```
DataFrame.isnull()
```

或者用命令:

```
DataFrame.notnull()
```

如果数据有缺失值, 则 isnull()方法的返回值为 True, 而 notnull()方法的返回值为 False。

2. 缺失值的填充

如果数据出现缺失值 (NaN), 与之计算的结果也会是 NaN, 因此要对缺失值进行填充。

其格式如下:

```
DataFrame.fillna(value=None, method=None, limit=None)
```

其中, value 是数值; method 的值有 backfill, bfill, pad, ffill, None, 默认是 None; limit 是填充次数。

3. 缺失值的删除

对于无法填充的缺失值，只能进行删除。

其格式如下：

```
DataFrame.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)
```

其中，axis=0 是 index 的行数据，axis=1 是 columns 的列数据。

当 how='any'时，则表示只要有一个缺失值就删除；当 how='all'时，则表示所在行或列全部无效才能删除。

4. 重复值的删除

如果数据出现重复可能会影响分析结果，因此就要删除这些重复值。

其格式如下：

```
DataFrame.drop_duplicates(subset=None, keep='first', inplace=False)
```

其中，subset 是列名。当 keep='first'时，删除重复项并保留第一次出现的项；当 inplace=False 时，则不修改原始数据。

6.6 DataFrame索引操作

在量化分析中，DataFrame 索引经常被用到。在 Tushare 股票历史数据中索引是 RangeIndex，在聚宽数据中索引是 DatetimeIndex，在 QUANTAXIS 数据中索引是 Index，因此我们会经常做不同索引项之间的转换。

在这里推荐使用 RangeIndex，这种默认的索引很容易进行数据处理，对于增加新列数也非常方便。但是对于图形显示要使用 DatetimeIndex，因为图形的纵坐标是日期或时间。

1. 修改索引

DataFrame 通过 set_index 方法可以设置单索引和复合索引。DataFrame 设置新索引的格式如下：

```
DataFrame.set_index(keys, drop=True, append=False, inplace=False, verify_integrity=False)
```

其中, `keys` 的值可以是序列类型, 也可以是调用者的一个列名, 即将某一列设为新数组的索引; `append` 为添加新索引。当 `drop` 为 `False`, `inplace` 为 `True` 时, 索引就会还原为列。

例如:

```
df1=data.set_index('date') #将 date 列设置为新索引
```

2. 重置索引

利用 `reset_index` 方法可以还原索引, 即重新变为默认的整型索引, 其格式如下:

```
DataFrame.reset_index(level=None, drop=False, inplace=False, col_level=0, col_fill='')
```

其中, `level` 控制具体要还原的那个等级的索引。如果 `drop` 为 `False`, 则索引列就会被还原为普通列, 否则会丢失。

例如:

```
data.reset_index()
```

见示例 6-15。

```
import pandas as pd
import tushare as ts
data = ts.get_k_data('600080',ktype='D')
print('import tushare as ts')
print('data = ts.get_k_data(\'600099\',ktype=\'D\')')
print(data.head(2))
print('type(data.index) ',type(data.index))
df1=data.set_index('date') #将 date 列设置为新索引
print('\ndf1=data.set_index(\'date\')')
print(df1.head(2))
print('type(df1.index) ',type(df1.index))
df2=df1.reset_index()
print('\ndf2=df1.reset_index()')
print(df2.head(2))
print('type(df2.index) ',type(df2.index))
```

程序运行结果：

```
import tushare as ts
data = ts.get_k_data('600099',ktype='D')
      date    open    close    high    low    volume    code
0 2016-05-13   9.792   9.981  10.031   9.792  32052.0  600080
1 2016-05-16  10.011  10.319  10.329   9.991  46384.0  600080
type(data.index)    <class 'pandas.core.indexes.range.RangeIndex'>

df1=data.set_index('date')
      date    open    close    high    low    volume    code
2016-05-13   9.792   9.981  10.031   9.792  32052.0  600080
2016-05-16  10.011  10.319  10.329   9.991  46384.0  600080
type(df1.index)     <class 'pandas.core.indexes.base.Index'>

df2=df1.reset_index()
      date    open    close    high    low    volume    code
0 2016-05-13   9.792   9.981  10.031   9.792  32052.0  600080
1 2016-05-16  10.011  10.319  10.329   9.991  46384.0  600080
type(df2.index)     <class 'pandas.core.indexes.range.RangeIndex'>
```

6.7 DataFrame数据的追加与合并

1. DataFrame 数据追加

如果两个 DataFrame 数据的列名相同，就可以将这两个 DataFrame 数据合并成一个。

格式如下：

```
DataFrame.append(other, ignore_index=False, verify_integrity=False,
sort=None)
```

其中，other 是其他的 DataFrame 数据、Series 数据、列表数据等；ignore_index 默认为 False，如果是 True，就不使用索引标签；verify_integrity 默认为 False，如果是 True，就提高 ValueError 和重复创建索引；sort=False 表示不排序。

例如:

```
df1.append(df2)
```

2. DataFrame 数据合并

合并两个 DataFrame 数据可以使用如下格式:

```
DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)
```

DataFrame.merge 参数种类及说明见表 6-6。

表 6-6 DataFrame.merge 参数说明

| 参数 | 说明 |
|-------------|---|
| right | 另一个 DataFrame 数据对象 |
| how | 数据融合的方法, 可选'left', 'right', 'outer', 'inner', 分别代表左、右、外在、内部, 默认为内部; left: 仅使用左边数据框的键, 类似于 SQL 左外连接; 保持按键顺序。 right: 仅使用右数据框中的键, 类似于 SQL 右外连接; 保持按键顺序。 outer: 使用两个数据框中的键的联合, 类似于 SQL 全外连接; 按字典顺序排序键。 inner: 使用来自两个数据框的键的交集, 类似于 SQL 内部联接; 保留左键的顺序 |
| on | 要加入的列名。在使用这个参数时要保证左表和右表对齐的那一列都有相同的列名 |
| left_on | 左表对齐的列, 可以是列名, 也可以是和 DataFrame 同样长度的 arrays |
| right_on | 右表对齐的列, 可以是列名, 也可以是和 DataFrame 同样长度的 arrays |
| left_index | Bool, 默认为 False, 使用索引从左边 DataFrame 连接键。如果是 MultiIndex, 则 key 在其他 DataFrame (索引或列数) 中必须匹配 |
| right_index | Bool, 默认为 False, 使用索引从右边 DataFrame 连接键 |
| sort | Bool, 默认为 False。DataFrame 连接键按字母排序的结果。如果为 False, 连接 key 的顺序就取决于连接关键字的类型 |
| suffixes | 字符串后缀并不适用于重叠列的元组, 默认值为 ('_x', '_y') |
| copy | Bool, 默认为 True。如果为 False, 就尽可能避免复制 |
| indicator | bool 或 str, 默认为 False。如果为 True, 就添加一个列, 输出的 DataFrame 称为_merge, 每一行的来源信息。如果为字符串, 列的每一行的信息就被添加到输出 DataFrame 和列中, 被命名为值的字符串 |
| validate | str, 可选。如果指定, 就检查是否指定类型的合并; one_to_one 或 1: 1: 检查合并键是否是独特的, 在左边和右边的数据集; one_to_many 或 1: m: 检查合并键是否是独特的数据集; many_to_one 或 m: 1: 检查合并键是否是唯一正确的数据集; many_to_many 或 m: m: 允许, 但不会导致检查 |

3. DataFrame 通过索引拼接列

内置的 join 方法是一种快速合并的方法，它以 index 作为对齐的列，可以把一个 DataFrame 数据列合并到另一个 DataFrame 数据中。

格式如下：

```
DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='',
sort=False)
```

join 方法中的参数与 merge 方法中的意义基本相同，只是 join 方法默认为左外连接，即 how=left。其中，other 是其他的 DataFrame 数据、Series 数据或列表；how 可选 left, right, outer, inner 等，默认值是 left。

索引拼接列的特点：

- (1) 默认按索引合并，可以合并相同或相似的索引，不管它们有没有重叠列。
- (2) 可以拼接多个 DataFrame 数据。
- (3) 可以拼接除索引外的其他列。
- (4) 拼接方式用参数“how”控制。
- (5) 通过“lsuffix=”和“rsuffix=”来区分相同列名的列。

在进行列拼接时，如果出现数值“NaN”，其主要原因就是两个 DataFrame 数据的索引不相同。

4. DataFrame 通过轴连接

pandas.concat 方法可以沿着一条轴将多个 DataFrame 数据堆叠到一起。concat 方法相当于数据库中的全连接，可以指定按某个轴进行连接，也可以指定连接的方式，如 join 方法（参数 outer 或 inner）。与数据库不同的是，concat 方法不会去重，要达到去重的效果可以使用 drop_duplicates 方法。

格式如下：

```
DataFrame=pandas.concat(objs, axis=0, join='outer', join_axes=None,
ignore_index=False, keys=None, levels=None, names=None,
verify_integrity=False, sort=None, copy=True)
```

其中，objs 是需要连接的 DataFrame 数据、Series 数据、列表或字典。axis 是需要合并连接的轴，其中 axis=0 是按行连接，axis=1 是按列连接。join 只有 inner 和

outer 可选。

pandas.concat 方法的特点：

- (1) 在作用于 Series 数据时，如果 axis=0，则类似于联合。如果 axis=1，则组成一个 DataFrame 数据。
- (2) 可以通过参数 join_axes=[] 指定自定义索引。
- (3) 可以通过参数 keys=[] 创建层次化索引。
- (4) 可以通过参数 ignore_index=True 重建索引。

6.8 DataFrame数据的保存和读取

由于 Pandas 是数据库分析专用库，因此其读取和保存（或写入）的接口函数或方法很多，见表 6-7。

表 6-7 Pandas 的读取和写入的函数或方法

| 读取函数或方法 | 写入函数或方法 |
|----------------|--------------|
| read_csv | to_csv |
| read_excel | to_excel |
| read_hdf | to_hdf |
| read_sql | to_sql |
| read_json | to_json |
| read_html | to_html |
| read_stata | to_stata |
| read_clipboard | to_clipboard |
| read_pickle | to_pickle |
| read_msgpack | to_msgpack |
| read_gbq | to_gbq |

下面介绍 CSV 和 Excel 文件的使用。

1. 保存 CSV

保存 CSV 的格式：

```
DataFrame.to_csv(path_or_buf=None, sep=',', na_rep='',
float_format=None, columns=None, header=True, index=True, index_label=None,
```

```
mode='w', encoding=None, compression=None, quoting=None, quotechar='\"',
line_terminator='\\n', chunksize=None, tupleize_cols=None, date_format=None,
doublequote=True, escapechar=None, decimal='\\.')

```

DataFrame .to_csv 参数种类及说明见表 6-8。

表 6-8 DataFrame.to_csv 参数说明

| 参数 | 说明 |
|-----------------|--|
| path_or_buf | 文件路径或文件对象，默认为 None。如果提供 None，则结果以字符串形式返回 |
| sep | character，默认为','，输出文件的字段分隔符 |
| na_rep | string，默认为'\\'，缺少数据表示 |
| float_format | string，默认为 None，格式字符串的浮点数 |
| columns | 列表，可选项，写入文件的列 |
| header | boolean 或字符串列表，默认为 True，写出列名。如果给出了字符串列表，则假定它是列名的别名 |
| index | boolean 默认为 True，写行名（索引） |
| index_label | string 或列表，或 False，默认为 None，索引列的列标签（如果需要）。如果没有给出并且 header 和 mdex 是 True，则使用索引名称。如果 DataFrame 使用 MultiIndex，则应该给出一个序列。如果 False 不打印索引名称的字段，就使用 index_label = False 在 R 中导入 |
| mode | str，Python 文件书写模式，默认为'w'模式 |
| encoding | string 可选项，表示要在输出文件中使用的编码的字符串。在 Python 2 中默认为 ascii，在 Python 3 中默认为 utf-8 |
| compression | string 可选项，表示要在输出文件中使用的压缩的字符串，允许的值是'gzip', 'bz2', 'xz'，仅当第一个参数是文件名 |
| line_terminator | string，默认为 '\\n'，输出文件中使用的换行符或字符序列 |

见示例 6-16。

```
#下载 tushare 基本数据
import tushare as ts
import pandas as pd
datapath='\\x\\data'
base=ts.get_stock_basics()
base.to_csv(datapath+'\\stock_base.csv' , encoding= 'gbk')

```

2. 读取 CSV

读取 CSV 的格式：


```
DataFrame=pandas.read_csv(filepath_or_buffer, sep=', ', delimiter=None,
header='infer', names=None, index_col=None, usecols=None, squeeze=False,
prefix=None, mangle_dupe_cols=True, dtype=None, engine=None,
converters=None, true_values=None, false_values=None,
skipinitialspace=False, skiprows=None, nrows=None, na_values=None,
keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True,
parse_dates=False, infer_datetime_format=False, keep_date_col=False,
date_parser=None, dayfirst=False, iterator=False, chunksize=None,
compression='infer', thousands=None, decimal=b'.', lineterminator=None,
quotechar='"', quoting=0, escapechar=None, comment=None, encoding=None,
dialect=None, tupleize_cols=None, error_bad_lines=True,
warn_bad_lines=True, skipfooter=0, doublequote=True,
delim_whitespace=False, low_memory=True, memory_map=False,
float_precision=None)
```

pandas.read_csv 参数种类及说明见表 6-9。

表 6-9 pandas.read_csv 参数说明

| 参数 | 说明 |
|--------------------|--|
| filepath_or_buffer | str, 文件名路径, 或者是 read()读取的文件缓存, 可以是 URL 类型, 包括 http, ftp, s3 和文件 |
| sep | str, 默认为',', 指定分隔符。如果不指定参数, 则会尝试使用逗号分隔。当分隔符长于一个字符时, 不是使用\s+而是使用 Python 的语法分析器, 并且忽略数据中的逗号 |
| delimiter | str, 默认为 None, 定义符, 备选分隔符 (如果指定该参数, 则 sep 参数失效) |
| delim_whitespace | boolean, default False, 指定空格 (如' ') 是否作为分隔符使用, 等效于设定 sep='\s+'。如果这个参数设定为 True, 则 delimiter 参数失效 |
| header | int or list of ints, 默认为'infer', 指定行数用来作为列名, 数据开始行数。如果文件中没有列名, 则默认为 0, 否则设置为 None。如果明确设定 header=0, 就会替换掉原来存在的列名。header 参数可以是一个 list。注意: 如果 skip_blank_lines=True, 那么 header 参数忽略注释行和空行, 所以 header=0 表示第一行数据而不是文件的第一行 |
| names | array-like, 默认为 None, 用于结果的列名列表。如果在数据文件中没有表头, 则需执行 header=None, 而且默认在列表中不能出现重复, 除非设定参数 mangle_dupe_cols=True |
| index_col | int or sequence or False, 默认为 None, 用作行索引的列编号或者列名。如果给定一个序列则有多个行索引。如果文件不规则, 行尾有分隔符, 则可以设定 index_col=False, 但 Pandas 不适用第一列作为行索引 |
| usecols | array-like, 默认为 None, 返回一个数据子集, 该列表中的值必须可以对应到文件中的位置 (数字可以对应到指定的列) 或者是字符串为文件中的列名, 使用这个参数可以加快加载速度并降低内存消耗 |

续表

| 参数 | 说明 |
|-----------------------|--|
| as_reccarray | boolean, 默认为 False, 该参数会从未版本中移除, 建议用 <code>pd.read_csv(...).to_records()</code> 替代, 并返回一个 NumPy 的 recarray 来替代 DataFrame。如果该参数设定为 True, 就会优先使用 <code>squeeze</code> 参数, 并且行索引不再可用, 索引的列也将被忽略 |
| squeeze | boolean, 默认为 False。如果文件值包含一列, 则返回一个 Series |
| prefix | str, 默认为 None。在没有列标题时, 给列添加前缀 |
| mangle_dupe_cols | boolean, 默认为 True, 重复的列, 将 'X'...'X' 表示为 'X.0'...'X.N'。如果设定为 False 就会将所有重名的列覆盖 |
| dtype | Type name or dict of column -> type, 默认为 None, 每列数据的数据类型 |
| engine | engine: {'c', 'python'}, optional, 使用的分析引擎, 可以选择 C 或者 Python。其中, C 为引擎快, 而 Python 的引擎功能更加完备 |
| converters | dict, 默认为 None, 列转换函数的字典, 其中 key 可以是列名也可以是列的序号 |
| true_values | list, 默认为 None, Values to consider as True |
| false_values | list, 默认为 None, Values to consider as False |
| skipinitialspace | boolean, 默认为 False, 忽略分隔符后的空白 (默认为 False, 即不忽略) |
| skiprows | list-like or integer, 默认为 None, 需要忽略的行数 (从文件开始处算起), 或需要跳过的行号列表 (从 0 开始) |
| skipfooter | int, default 0, 从文件尾部开始忽略 (c 引擎不支持) |
| skip_footer | int, default 0, 建议使用 skipfooter 代替, 功能一样 |
| nrows | int, default None, 需要读取的行数 (从文件开始处算起) |
| na_values | scalar, str, list-like, or dict, 默认为 None, 一组用于替换 NA/NaN 的值。如果需要传递参数, 就制定特定列的空值, 默认为 'I.#IND', 'I.#QNaN', 'N/A', 'NA', 'NULL', 'NaN', 'nan' |
| keep_default_na | bool, default True, 如果指定 na_values 参数, 并且 keep_default_na=False, 那么默认的 NaN 就会被覆盖, 否则需要添加 |
| na_filter | boolean, 默认为 True, 是否检查丢失值 (空字符串或者空值)。对于大文件来说, 数据集中没有空值, 设定 na_filter=False 可以提升读取速度 |
| verbose | boolean, 默认为 False, 是否打印各种解析器的输出信息, 如 “非数值列中缺失值的数量” 等 |
| skip_blank_lines | boolean, default True, 如果为 True, 则跳过空行; 否则记为 NaN |
| parse_dates | boolean or list of ints or names or list of lists or dict, 默认为 Falseboolean. True -> 解析索引 |
| infer_datetime_format | boolean, 默认为 False, 如果设定为 True, 并且 parse_dates 可用, 那么 Pandas 就尝试转换为日期类型; 如果可以转换且转换方法能解析, 在某些情况下就会快 5~10 倍 |



续表

| 参数 | 说明 |
|-----------------|--|
| keep_date_col | boolean, 默认为 False。如果连接多列解析日期, 则保持参与连接的列 |
| date_parser | function, 默认为 None, 用于解析日期的函数, 默认使用 dateutil.parser.parser 来做转换 |
| dayfirst | boolean, 默认为 False, DD/MM 格式的日期类型 |
| iterator | boolean, 默认为 False, 返回一个 TextFileReader 对象, 以便逐块处理文件 |
| chunksize | int, 默认为 None, 文件块的大小, See IO Tools docs for more information on iterator and chunksize |
| compression | {'infer', 'gzip', 'bz2', 'zip', 'xz', None}, 默认为 'infer', 直接使用磁盘上的压缩文件。如果使用 infer 参数, 则使用 gzip, bz2, zip 或者解压文件名中以 .gz, .bz2, .zip, .xz 为后缀的文件, 否则无法解压。如果使用 zip, 那么 zip 包就必须只包含一个文件; 如果设置为 None 则不解压 |
| thousands | str, 默认为 None, 千分位分割符, 如 “,” 或者 “.” |
| decimal | str, 默认为 “.”, 字符中的小数点 (如欧洲数据使用 “,”) |
| float_precision | 字符串, 默认没有指定, C 解析器使用浮点值, 普通的选项没有转换器 |
| lineterminator | str (length 1), 默认为 None, 行分割符, 只在 C 解析器下使用 |
| quotechar | str (length 1), optional, 引号, 用作标识开始和解释的字符, 引号内的分割符将被忽略 |
| quoting | int or csv.QUOTE_* instance, 默认为 0, 控制 CSV 中的引号常量, 可选 QUOTE_MINIMAL (0), QUOTE_ALL (1), QUOTE_NONNUMERIC (2) 或 QUOTE_NONE (3) |
| doublequote | boolean, 默认为 True, 双引号, 当单引号已经被定义, 并且 quoting 参数不是 QUOTE_NONE 时, 则使用双引号表示引号内的元素并作为一个元素使用 |
| escapechar | str (length 1), 默认为 None, 当 quoting 为 QUOTE_NONE 时, 则指定一个字符使得不受分隔符的限制 |
| comment | str, 默认为 None, 标识着多余的行不被解析。如果该字符出现在行首, 则这一行就被全部忽略 |
| encoding | str, 默认为 None, 指定字符集类型, 通常指定为 'utf-8'. List of Python standard encodings |
| dialect | str or csv.Dialect instance, 默认为 None。如果没有指定特定的语言, 当 sep 大于一个字符时就可以忽略 |
| tupleize_cols | boolean, 默认为 False, 离开元组在列的列表 (默认是将多索引的列) |
| error_bad_lines | boolean, 默认为 True。如果一行包含太多的列, 那么就默认不会返回 DataFrame; 如果设置成 False, 那么就改行剔除 (只能在 C 解析器下使用) |

续表

| 参数 | 说明 |
|----------------|---|
| warn_bad_lines | boolean, 默认为 True。如果 error_bad_lines=False 且 warn_bad_lines=True, 那么所有的 bad lines 就会被输出 (只能在 C 解析器下使用) |
| low_memory | boolean, 默认为 True, 分块加载到内存, 再低内存消耗中解析, 但是可能会出现类型混淆。如果想确保类型不被混淆则需要设置为 False, 或者使用 dtype 参数指定类型。注意: 使用 chunksize 或 iterator 参数分块读入会将整个文件读入到一个 Dataframe 中, 从而忽略了类型 (只能在 C 解析器中有效) |
| buffer_lines | int, 默认为 None, 不推荐使用, 因为这个参数在未来版本中将会被移除, 且他的值在解析器中不推荐使用 |
| compact_ints | boolean, 默认为 False, 不推荐使用, 这个参数在未来版本中将会被移除。 如果设置 compact_ints=True, 那么任何有整数类型构成的列都会被按照最小的整数类型存储, 是否有符号将取决于 use_unsigned 参数 |
| use_unsigned | boolean, 默认为 False, 不推荐使用, 这个参数在未来版本中将会被移除。 如果整数列被压缩 (i.e. compact_ints=True), 就需要指定被压缩的列是否有符号 |
| memory_map | boolean, 默认为 False。如果使用的文件在内存内, 那么就直接使用 map 文件, 这样可以避免文件再次进行 IO 操作 |

见示例 6-17。

```
import pandas as pd
#读取 xbdata 目录双色球开奖数据
def get_ssq():
    base=pd.read_csv('\\xbdata\\cp\\ssq.csv' , encoding= 'gbk')
    return base
print(get_ssq().head(5))
```

3. 保存 Excel

保存 Excel 格式:

```
DataFrame.to_excel(excel_writer, sheet_name='Sheet1', na_rep='',
float_format=None, columns=None, header=True, index=True, index_label=None,
startrow=0, startcol=0, engine=None, merge_cells=True, encoding=None,
inf_rep='inf', verbose=True, freeze_panes=None)
```

DataFrame.to_excel 方法的参数种类及说明见表 6-10。

表 6-10 DataFrame.to_excel 方法的参数种类及说明

| 参数 | 说明 |
|--------------|--|
| excel_writer | 写入的目标 Excel 文件，如文件路径、ExcelWriter 对象 |
| sheet_name | 被写入的 Sheet 名称，string 类型，默认为'sheet1' |
| na_rep | 缺失值表示，string 类型 |
| header | 是否写表头信息，布尔或 list of string 类型，默认为 True |
| index | 是否写行号，布尔类型，默认为 True |
| encoding | 指定写入编码，string 类型 |

见示例 6-18。

```
import pandas as pd
import tushare as ts
df = ts.get_k_data('600080', ktype='D')
df1=df.head()
df2=df.tail()
writer = pd.ExcelWriter('output.xlsx')
df1.to_excel(writer, 'Sheet1')
df2.to_excel(writer, 'Sheet2')
writer.save()
```

4. 读取 Excel

读取 Excel 的格式：

```
DataFrame=pandas.read_excel(io, sheet_name=0, header=0, names=None,
index_col=None, usecols=None, squeeze=False, dtype=None, engine=None,
converters=None, true_values=None, false_values=None, skiprows=None,
nrows=None, na_values=None, parse_dates=False, date_parser=None,
thousands=None, comment=None, skipfooter=0, convert_float=True,
**kwds) [source]
```

pandas.read_excel 函数的参数种类及说明见表 6-11。

表 6-11 pandas.read_excel 函数的参数种类及说明

| 参数 | 说明 |
|-----------|--|
| io | Excel 文件，如文件路径、文件网址、file-like 对象、xlrd workbook |
| sheetname | 返回指定的 Sheet，参数可以是字符串（Sheet 名）、整型（Sheet 索引）、list（元素为字符串和整型、返回字典{'key':'sheet'}）、none（返回字典，全部 Sheet） |

续表

| 参数 | 说明 |
|----------|--|
| header | 指定数据表的表头，参数可以是 int，list of ints 即为索引行数为表 |
| names | 返回指定 name 的列，参数为 array-like 对象 |
| encoding | 关键字参数，指定以何种编码读取 |

见示例 6-19:

```
import pandas as pd
rd1=pd.read_excel('output.xlsx','Sheet1')
rd2=pd.read_excel('output.xlsx','Sheet2')
print(rd1)
print(rd2)
```

6.9 DataFrame运算

(1) DataFrame 列数据的运算。

DataFrame 列数据支持+，-，*，/，**等运算，如 `x=df.close*2`。

(2) DataFrame 的统计方法。

DataFrame 的统计方法非常丰富，常用的统计方法及功能见表 6-12。

表 6-12 DataFrame 常用的统计方法

| 方法 | 功能 |
|--|--------------------------|
| DataFrame.abs() | 返回绝对值 |
| DataFrame.all([axis, bool_only, skipna]) | 返回所有元素是否为 True |
| DataFrame.any([axis, bool_only, skipna]) | 如果一个（或多个）元素是正确的，则返回 True |
| DataFrame.clip([lower, upper, axis]) | 调整值为输入阈值（s） |
| DataFrame.clip_lower(threshold[, axis]) | 调整值低于给定的阈值 |
| DataFrame.clip_upper(threshold[, axis]) | 调整值高于给定的阈值 |
| DataFrame.corr([method, min_periods]) | 返回本数据框成对列的相关性系数 |
| DataFrame.corrwith(other[, axis, drop]) | 返回不同数据框的相关性 |
| DataFrame.count([axis, level, numeric_only]) | 返回非空元素的个数 |
| DataFrame.cov([min_periods]) | 计算协方差 |
| DataFrame.cummax([axis, skipna]) | 返回在 DataFrame 中累积最大或系列的轴 |



续表

| 方法 | 功能 |
|---|----------------------------|
| DataFrame.cummin([axis, skipna]) | 返回一个 DataFrame，返回所请求累积最低的轴 |
| DataFrame.cumprod([axis, skipna]) | 返回累积 |
| DataFrame.cumsum([axis, skipna]) | 返回累和 |
| DataFrame.describe([percentiles, include, ...]) | 整体描述数据框 |
| DataFrame.diff([periods, axis]) | 返回第一个离散元素的差异 |
| DataFrame.eval(expr[, inplace]) | 评估一个字符串描述 DataFrame 列的操作 |
| DataFrame.kurt([axis, skipna, level, ...]) | 返回无偏峰度 |
| DataFrame.mad([axis, skipna, level]) | 返回偏差 |
| DataFrame.max([axis, skipna, level, ...]) | 返回最大值 |
| DataFrame.mean([axis, skipna, level, ...]) | 返回均值 |
| DataFrame.median([axis, skipna, level, ...]) | 返回中位数 |
| DataFrame.min([axis, skipna, level, ...]) | 返回最小值 |
| DataFrame.mode([axis, numeric_only]) | 返回众数 |
| DataFrame.pct_change([periods, fill_method]) | 返回百分比变化 |
| DataFrame.prod([axis, skipna, level, ...]) | 返回连乘积 |
| DataFrame.quantile([q, axis, numeric_only]) | 返回分位数 |
| DataFrame.rank([axis, method, numeric_only]) | 返回数字的排序 |
| DataFrame.round([decimals]) | 返回一个 DataFrame 数量可变的小数位 |
| DataFrame.sem([axis, skipna, level, ddof]) | 返回无偏标准误 |
| DataFrame.skew([axis, skipna, level, ...]) | 返回无偏偏度 |
| DataFrame.sum([axis, skipna, level, ...]) | 求和 |
| DataFrame.std([axis, skipna, level, ddof]) | 返回标准误差 |
| DataFrame.var([axis, skipna, level, ddof]) | 返回无偏误差 |

下面看一个统计示例 6-20:

```
import tushare as ts
df1 = ts.get_k_data('000776', ktype='D').head()
print(df1)
print('求各列的平均值')
df2=df1.mean(axis=0, skipna=False)          #求各列的平均值，考虑 na 的存在
print(df2)
print('求各列的最大值')
df3=df1.max(axis=0, skipna=False)           #求各列的最大值，考虑 na 的存在
print(df3)
```

程序运行结果:

```

      date    open  close   high    low   volume   code
0  2016-11-07  17.194  16.994  17.241  16.928  480997.0  000776
1  2016-11-08  17.184  17.089  17.336  17.013  411784.0  000776
2  2016-11-09  17.051  16.795  17.156  16.510  643966.0  000776
3  2016-11-10  17.032  17.298  17.564  17.032  872295.0  000776
4  2016-11-11  17.374  17.840  18.011  17.175  1189926.0  000776
求各列的平均值
open              17.1670
close             17.2032
high              17.4616
low               16.9316
volume            719793.6000
dtype: float64
求各列的最大值
date              2016-11-11
open              17.374
close             17.84
high              18.011
low               17.175
volume            1.18993e+06
code              000776
dtype: object

```

6.10 DataFrame数据画线

Pandas 具有简单绘制数据线的功能, 可以绘制折线图、条形图等。绘制简单的图形也可以直接使用 DataFrame.plot 方法。

DataFrame.plot 方法绘图的函数格式如下:

```

DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False,
sharex=None, sharey=False, layout=None, figsize=None, use_index=True,
title=None, grid=None, legend=True, style=None, logx=False, logy=False,
loglog=False, xticks=None, yticks=None, xlim=None, ylim=None, rot=None,

```



```
fontsize=None, colormap=None, table=False, yerr=None, xerr=None,
secondary_y=False, sort_columns=False, **kwds)
```

DataFrame.plot 的主要参数及说明见表 6-13。

表 6-13 DataFrame.plot 的主要参数及说明

| 参数 | 说明 |
|-----------|--|
| x | label 或 position, 默认为 None, 指数据框列的标签或位置参数 |
| y | label 或 position, 默认为 None |
| kind | Str, 图形形状。 'line': line plot (default), 折线图 'bar': vertical bar plot, 条形图 'barh': horizontal bar plot, 横向条形图 'hist': histogram, 柱状图 'box': boxplot, 箱线图 'kde': Kernel Density Estimation plot, Kernel 的密度估计图, 主要是对柱状图添加 Kernel 概率密度线 'density': 同'kde' 'area': area plot 'pie': pie plot#饼图 'scatter': scatter plot#散点图 'hexbin': hexbin plot |
| ax | matplotlibaxesobject, 默认为 None, 一个图片切成不同片段, 即子图对象 |
| subplots | boolean, 默认为 False, 判断图片中是否有子图 |
| sharex | boolean, 默认为 True;ifaxisNoneelseFalse, 如果有子图, 则子图就共 x 轴刻度、标签 |
| sharey | boolean, 默认为 False, 如果有子图, 则子图就共 y 轴刻度、标签 |
| layout | tuple(optional), 子图的行列布局 |
| figsize | atuple(width, height)inches, 图片尺寸大小 |
| use_index | boolean, 默认为 True, 默认用索引做 x 轴 |
| title | String, 图片的标题用字符串 |
| grid | boolean, 默认为 None(matlabstyledefault), 图片是否有网格 |
| legend | False/True/'reverse', 子图的图例 |
| style | listordict, 对每列折线图设置线的类型 |
| logx | boolean, 默认为 False, 设置 x 轴刻度是否取对数 |
| logy | boolean, 默认为 False |
| loglog | boolean, 默认为 False, 同时设置 x 轴, y 轴刻度是否取对数 |

续表

| 参数 | 说明 |
|----------|--|
| xticks | sequence, 设置 x 轴刻度值, 序列形式 (如列表) |
| yticks | sequence, 设置 y 轴刻度值, 序列形式 (如列表) |
| xlim | 2-tuple/list 设置坐标轴的范围, 列表或元组形式 |
| ylim | 2-tuple/list |
| rot | int, 默认为 None, 设置轴标签 (轴刻度) 的显示旋转度数 |
| fontsize | int, 默认为 None, 设置轴刻度的字体大小 |
| colormap | str 或 matplotlib colormap object, 默认为 None, 设置图的区域颜色 |

见示例 6-21。

```
import tushare as ts
df = ts.get_k_data('000776', ktype='D').head(10)
df.open.plot.line(legend=True)
df.low.plot.bar(legend=True)
```

6.11 仿通达信大智慧公式指标KDJ

本节学习将通达信 KDJ 指标移植到 Python 系统中的例子, 这是对我们学习本章知识的综合训练。

股票公式也称股票指标, 或股票指标公式。股票软件公式语法最早是分析家专业版软件上的公式系统语法, 由于使用它编写股票新指标比较方便, 所以逐步成为主流指标公式语言。之后出现的飞狐交易师、通达信、同花顺等都是使用的分析家公式语法, 因此它们除了增加一些各自独特的新语法和新公式的基础函数外, 指标公式语法大致相同。

在 Python 中, 新写的公式在 Python 语法上称为自定义函数。在自定义函数中使用的大量搭建公式的函数, 称为公式基础函数。例如, VMA(), CMA()等函数。

在一般股票软件中, 不区分大小写, 如 close 同 CLOSE。其中, 默认 C 或 CLOSE 为收盘价; L 或 LOW 为最低价; O 或 OPEN 为开盘价; H 或 HIGH 为最高价; V 或 VOL 为成交量; AMO 或 AMOUNT 为成交额。此外, 行情还包括指数行情, 其中包含上涨家数、下跌家数等信息, 还包含股票软件提供的一些财务数据, 如流通盘、每股收益、市盈率、市净率、除权信息等。这些暂不在我们现在学习讨论的范围内,

以后会逐步讲解这些知识和程序分析。

通达信 KDJ 指标的公式，见图 6-1。



图 6-1 通达信 KDJ 指标的公式

将股票公式转为 Python 代码，我们可以采用“自顶向下，逐步求精”的面向过程的设计思路。过程化和模块化，比较符合股票软件的公式语法。

1. 构造基础行情数据

首先，要构造股票行情的基础数据，如 C 或 CLOSE 为收盘价，L 或 LOW 为最低价，O 或 OPEN 为开盘价，H 或 HIGH 为最高价等。

这些数据都是以日期时间为索引的序列，刚好符合 Pandas 的 Series 类型，因此将它们组合到一起可以形成 DataFrame 类型。

为了方便公式处理，不同来源的行情数据都要转化为类似于 Tushare 数据的 DataFrame 结构。转化后的数据主要包括 index, date, close, open, low, high 六个列名，假设转化后的 DataFrame 名为 mydf。

Python 的变量通过等号“=”赋值，变量得到的是对象地址指针，这个对象变量指向原始数据地址。Series 类型的数据就是对象数据，我们可以利用对象变量赋值的特性构造 Python 中的股票基础数据。

构造程序代码如下：

```
##仿通达信系统公式初始化
#mydf 是一个股票数据序列，将股票数据赋值给 mydf
```

```
#mydf=data.copy()#建立新的数据,防止破坏原始数据
CLOSE=mydf['close']
LOW=mydf['low']
HIGH=mydf['high']
OPEN=mydf['open']
VOL=mydf['volume']
C=mydf['close']
L=mydf['low']
H=mydf['high']
O=mydf['open']
V=mydf['volume']
```

2. 编写股票公式的基础函数

股票公式的基础函数很多,我们不一一讨论,下面给出常用的几个基础函数。
基础函数代码如下:

```
def MA(Series, N):
    return pd.Series.rolling(Series, N).mean()

def SMA(Series, N, M=1):
    ret = []
    i = 1
    length = len(Series)
    #跳过 x 中前面几个 nan 值
    while i < length:
        if np.isnan(Series.iloc[i]):
            i += 1
        else:
            break
    preY = Series.iloc[i] #Y'
    ret.append(preY)
    while i < length:
        Y = (M * Series.iloc[i] + (N - M) * preY) / float(N)
        ret.append(Y)
        preY = Y
        i += 1
```



```

return pd.Series(ret, index=Series.tail(len(ret)).index)

def HHV(Series, N):
    return pd.Series(Series).rolling(N).max()

def LLV(Series, N):
    return pd.Series(Series).rolling(N).min()

```

3. 通达信公式转为 Python 代码

通达信公式转为 Python 代码的过程如下：

- (1) “:=” 为赋值语句，用程序将 “:=” 替换为 Python 的赋值命令 “=”。
- (2) “:” 为公式的赋值带输出画线命令，将 “:” 替换为 “=”，并将 “:” 前的指标线输出变量顺序写到 Python 函数的 “return” 返回参数中。

- (3) 全部命令转为英文大写。
- (4) 删除绘图格式命令。
- (5) 删除每行末的分号 “;”。
- (6) 参数可写到函数参数表中，如 “def KDJ(N=9, M1=3, M2=3):”。

例如，通达信 KDJ 指标公式描述如下。

参数表 “N:=9, M1:=3, M2:=3”。

```

RSV:=(CLOSE-LLV(LOW,N))/(HHV(HIGH,N)-LLV(LOW,N))*100;
K:SMA(RSV,M1,1);
D:SMA(K,M2,1);
J:3*K-2*D;

```

将上面的股票公式转换成 Python 代码如下：

```

def KDJ(N=9, M1=3, M2=3):
    RSV = (CLOSE - LLV(LOW, N)) / (HHV(HIGH, N) - LLV(LOW, N)) * 100
    K = SMA(RSV, M1, 1)
    D = SMA(K, M2, 1)
    J = 3*K-2*D
    return K, D, J

```

4. 使用 KDJ 函数

在 Python 中的代码如下：

```
“K,D,J=KDJ()” 或 “K,D,J=KDJ(9,3,3)”。
```

得到 K, D, J 的列表数据。

5. 将 K,D,J 合成到 mydf 中

我们可以用 join()方法将 K,D,J 合成到 mydf 中。K 是列表，故需要增加列名，并转化为 Series 数据。

```
mydf = mydf.join(pd.Series( K,name='K'))#增加 K 到 mydf 中
mydf = mydf.join(pd.Series( D,name='D'))#增加 D 到 mydf 中
mydf = mydf.join(pd.Series( J,name='J'))#增加 J 到 mydf 中
```

6. 显示 KDJ 指标线

```
#下面是绘线语句
mydf.S80.plot.line()
mydf.X6.plot.line()
mydf.K.plot.line(legend=True)
mydf.D.plot.line(legend=True)
mydf.J.plot.line(legend=True)
```

最后给出示例 6-22 的全部代码。

```
'''
仿通达信大智慧公式指标 KDJ
'''
import numpy as np
import pandas as pd
import tushare as ts
#平均函数
def MA(Series, N):
    return pd.Series.rolling(Series, N).mean()
def SMA(Series, N, M=1):
    ret = []
    i = 1
```

```

length = len(Series)
#跳过X中前面几个 nan 值
while i< length:
    if np.isnan(Series.iloc[i]):
        i += 1
    else:
        break
preY = Series.iloc[i] #Y'
ret.append(preY)
while i< length:
    Y = (M * Series.iloc[i] + (N - M) * preY) / float(N)
    ret.append(Y)
    preY = Y
    i += 1

return pd.Series(ret, index=Series.tail(len(ret)).index)
def HHV(Series, N):
    return pd.Series(Series).rolling(N).max()
def LLV(Series, N):
    return pd.Series(Series).rolling(N).min()
data = ts.get_k_data('600099',ktype='D')
##仿通达信系统公式初始化
#mydf 是一个股票数据序列，将股票数据赋值给 mydf
mydf=data.copy() #建立新的数据，防止破坏原始数据
CLOSE=mydf['close']
LOW=mydf['low']
HIGH=mydf['high']
OPEN=mydf['open']
VOL=mydf['volume']
C=mydf['close']
L=mydf['low']
H=mydf['high']
O=mydf['open']
V=mydf['volume']
def KDJ(N=9, M1=3, M2=3):
    RSV = (CLOSE - LLV(LOW, N)) / (HHV(HIGH, N) - LLV(LOW, N)) * 100
    K = SMA(RSV,M1,1)
    D = SMA(K,M2,1)

```

```

J = 3*K-2*D
return K, D, J

K,D,J=KDJ()
mydf = mydf.join(pd.Series( K,name='K')) #增加 K 到 mydf 中
mydf = mydf.join(pd.Series( D,name='D')) #增加 D 到 mydf 中
mydf = mydf.join(pd.Series( J,name='J')) #增加 J 到 mydf 中
mydf['S80']=80 #增加上轨 80 迹线
mydf['X20']=20 #增加下轨 20 迹线
mydf=mydf.tail(100) #显示最后 100 条数据线
#下面是绘线语句
mydf.S80.plot.line()
mydf.X20.plot.line()
mydf.K.plot.line(legend=True)
mydf.D.plot.line(legend=True)
mydf.J.plot.line(legend=True)

```

程序运行结果，见图 6-2。

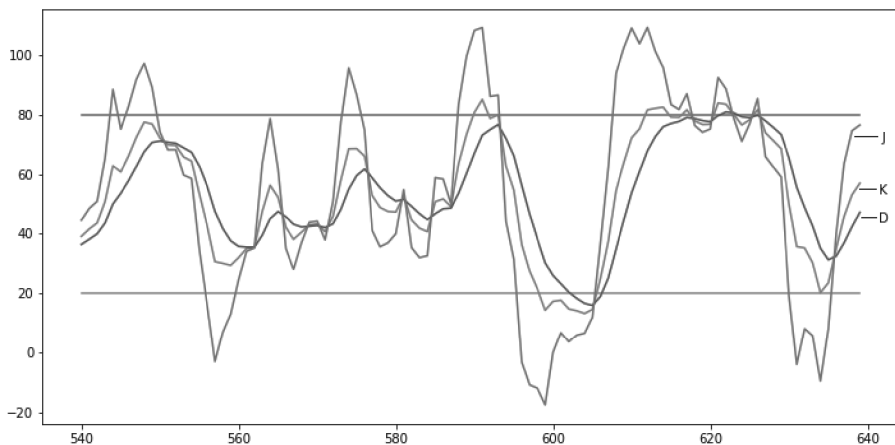


图 6-2 程序运行结果

本节是我们对这章学习的一个综合练习，同时也涉及了小白量化投资平台的一些基础功能。

7

第 7 章

Matplotlib 模块

Matplotlib 是 Python 最重要的绘图库。为了绘图方便，Matplotlib 的 Pyplot 模块提供了类似于 MATLAB 的界面。高级用户可以通过面向对象的界面，或类似于 MATLAB 的一组函数来完成控制线条样式、字体属性、轴属性等的设置。Matplotlib 是可用于 Python 脚本的图形用户界面的工具包，它可以让简单的事情变得更简单，让无法实现的事情变得有可能实现。它仅用几行代码就可以生成绘图、直方图、功率谱、条形图、错误图、散点图等，还能方便地将这些图作为绘图控件嵌入 GUI 应用程序中。

Matplotlib 在金融分析中应用非常广泛，也是量化投资分析的重要内容。本章重点讲解绘图部分——matplotlib.pyplot。

7.1 Matplotlib的使用

1. Matplotlib 的安装和输出

1) Matplotlib 的安装

Matplotlib 的安装命令如下：

```
pip install matplotlib
```

Matplotlib 库的装载命令如下：



```
import matplotlib
```

Pyplot 模块是 Matplotlib 的绘图子库，我们通常使用别名“plt”。

```
import matplotlib.pyplot as plt
```

2) Matplotlib 选择输出后端

本节仅适合 Matplotlib ver2.0.2 版本。为了支持所有的使用情形，Matplotlib 能够产生不同的输出，而每一个输出的能力叫作一个后端。其前端是用户需要处理的代码，如绘图代码；后端是生成视图窗口。Matplotlib 把不同的使用情形和输出格式作为目标，而使用者利用 Matplotlib 在 Python Shell 上的交互，即输入命令就能弹出绘图窗口。另外，也可以将 Matplotlib 嵌入到其他的用户图形接口中来编写丰富的应用程序。

这里有两种不同类型的后端：

(1) 用户接口后端，又称交互式后端，如 GTK3Agg, GTK3Cairo, macOS, nbAgg, Qt4Agg, Qt4Cairo, Qt5Agg, Qt5Cairo, TkAgg, TkCairo, WebAgg, WX, WXAgg, WXCairo。

(2) 硬拷贝后端，又称非交互式后端，用来生成图像文件，如 agg, cairo, pdf, pgf, ps, svg, template。

设置后端命令格式：

```
matplotlib.use(arg, warn=False, force=True)
```

设置后端命令的参数及说明见表 7-1。

表 7-1 matplotlib.use 参数说明

| 参数 | 说明 |
|-------|--|
| arg | 字符串，标准的后端名字。 互动后端：GTK3Agg, GTK3Cairo, macOS, nbAgg, Qt4Agg, Qt4Cairo, Qt5Agg, Qt5Cairo, TkAgg, TkCairo, WebAgg, WX, WXAgg, WXCairo。 非交互式的后端：agg, cairo, pdf, pgf, ps, svg, template。 或字符串的形式：module://my.module.name。 注：标准后端名称不区分大小写 |
| Warn | 布尔型，可选择的，默认值为 False。如果是 True，则检查系统导入是否与设置后端一致 |
| force | 布尔型，可选择的，默认为 True，即如果选择了一个交互式的后端，那么切换后端就会出现错误信息“ImportError” |

后端字符串及说明见表 7-2。

表 7-2 后端字符串说明

| 后端（不区分大小写） | 说明 |
|------------|---|
| GTKAgg | Agg 渲染器输出到 GTK 绘图面板（需要 PyGTK） |
| GTK | GDK 渲染器输出到 GTK 绘图面板（需要 PyGTK） |
| GTKCairo | Cairo 渲染器输出到 GTK 绘图面板（需要 PyGTK） |
| WXAgg | Agg 渲染器输出到 wxWidgets 绘图面板（需要 wxPython） |
| WX | 原生 wxWidgets 绘图输出到 wxWidgets 绘图面板（需要 wxPython） |
| TkAgg | Agg 渲染器输出到 Tk 绘图面板（需要 tkinter） |
| Qt5Agg | Agg 渲染器输出到 Qt5 绘图面板（需要 PyQt5） |
| Qt4Agg | Agg 渲染器输出到 Qt4 绘图面板（需要 PyQt4） |
| FLTKAgg | Agg 渲染器输出到 FLTK 绘图面板（需要 pyFLTK，使用不是很广，可以使用 TKAgg, GTKAgg, WXAgg, QT4Agg 替代） |
| macosx | Cocoa 渲染器在 osx 上 |

本书使用后端为 TkAgg，即 Agg 渲染器输出到 Tkinter 绘图面板。

3) Matplotlib 选择 Tkinter 后端

本书主要适用于 Tkinter 作为量化分析框架窗口，先介绍 Matplotlib 选择 Tkinter 后端输出图形的过程。

(1) 需要加载的几行模块命令如下：

```
import tkinter
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2TkAgg
```

(2) 在绘图前先使用如下命令，选择 Tkinter 为图形输出后端。

```
matplotlib.use('TkAgg')
```

(3) 设置 fig 绘图窗口，并在窗口中使用 plt 绘图。

```
fig = plt.figure()
```

(4) 建立 Tkinter 窗口。

```
root = tkinter.Tk()
```

(5) 设置一个 Tkinter 绘图区 “canvas”。

```
canvas =FigureCanvasTkAgg(fig, master=root)    #设置 tkinter 绘图区
canvas.draw()                                #绘制 plt 图形
```

(6) 可以选择使用 plt 图形导航工具条 NavigationToolbar。

```
toolbar =NavigationToolbar2TkAgg(canvas, root) #plt 图形导航工具条
toolbar.update()                             #显示 plt 图形导航工具条
```

(7) 在 Tkinter 窗口中放置 canvas 并显示。

```
canvas._tkcanvas.pack(side=tkinter.TOP, fill=tkinter.BOTH, expand=1)
```

(8) 运行 Tkinter 的 mainloop()主循环命令。

```
tkinter.mainloop()
```

下面是在 Tkinter 中显示 matplotlib.pyplot 图形的全部过程，见示例 7-1。

```
import tushare as ts
import tkinter
import matplotlib.pyplot as plt
#from matplotlib.backends.backend_tkagg import FigureCanvasTk,
NavigationToolbar2Tk                                #matplotlib 2.0.2
from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg, NavigationToolbar2Tk)        ##matplotlib 3.0.2
#matplotlib.use('TkAgg')                            #只有 matplotlib 2.0.2 需要，高版本可不用设置
plt.rcParams['font.sans-serif']=['SimHei']          #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False           #用来正常显示负号
df1 = ts.get_k_data('600080',ktype='D')
root = tkinter.Tk()
root.title="在 tkinter 中显示 matplotlib.pyplot 图形"
fig = plt.figure(1)
ax1 = plt.subplot2grid((3,3), (0,0), colspan=3)
ax2 = plt.subplot2grid((3,3), (1,0), colspan=2)
ax3 = plt.subplot2grid((3,3), (1, 2), rowspan=2)
ax4 = plt.subplot2grid((3,3), (2, 0))
ax5 = plt.subplot2grid((3,3), (2, 1))
plt.suptitle("在 tkinter 中显示 matplotlib.pyplot 图形")
```

```

ax1.plot(df1.close)           #在 ax1 中画 close 线
ax2.plot(df1.open)           #在 ax2 中画 open 线
ax3.plot(df1.volume)         #在 ax3 中画 volume 线
ax4.plot(df1.low)            #在 ax4 中画 low 线
ax5.plot(df1.high)           #在 ax5 中画 high 线
canvas =FigureCanvasTkAgg(fig, master=root)      #设置 tkinter 绘图区
canvas.draw()                 #绘制 plt 图形
#toolbar =NavigationToolbar2TkAgg(canvas, root)   #plt 图形导航工具条
matplotlib 2.0.2
    toolbar =NavigationToolbar2Tk(canvas, root)    #plt 图形导航工具条
matplotlib 3.0.2
    toolbar.update()           #显示 plt 图形导航工具条
    canvas._tkcanvas.pack(side=tkinter.TOP, fill=tkinter.BOTH, expand=1)
    plt.close()                #关窗口
    tkinter.mainloop()

```

程序运行结果见图 7-1。

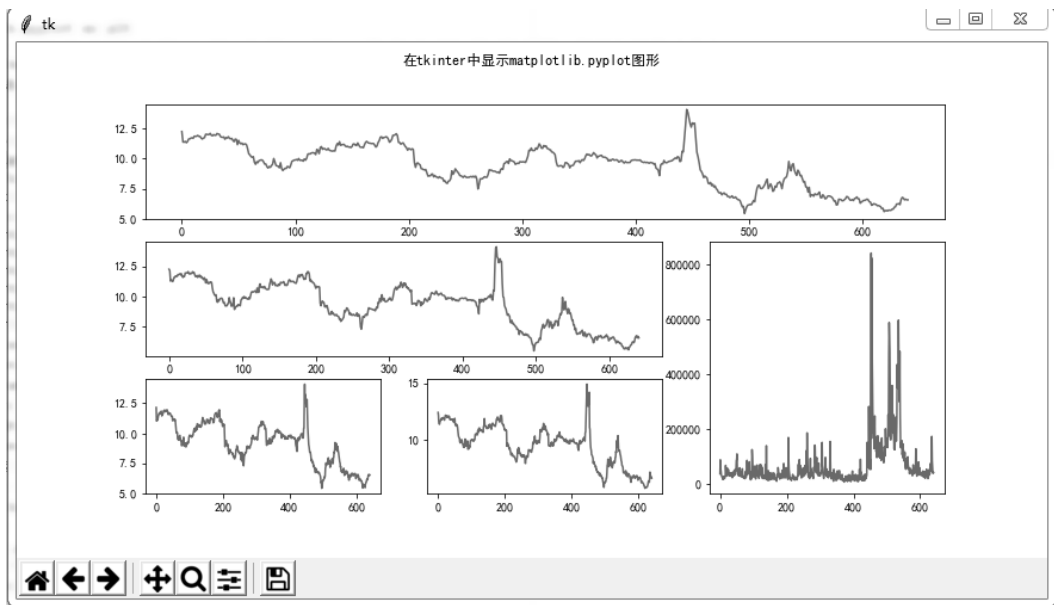


图 7-1 程序示例运行结果



注意

如果程序运行错误，就需要先在 Windows CMD 窗口下运行命令 `pip install tushare`。

2. Matplotlib 的画图流程

`matplotlib.pyplot` 是 Matplotlib 的一个基本绘图接口，它提供了一个类似于 MATLAB 绘图的方法。如果读者使用的是 Spyder 软件，则需要先进行设置使软件能显示 Pyplot 窗口。

Spyder 软件的设置：

在“Tools—Preferences—IPython console—Graphics—Backend”菜单项选择“Automatic”。如图 7-2 所示。

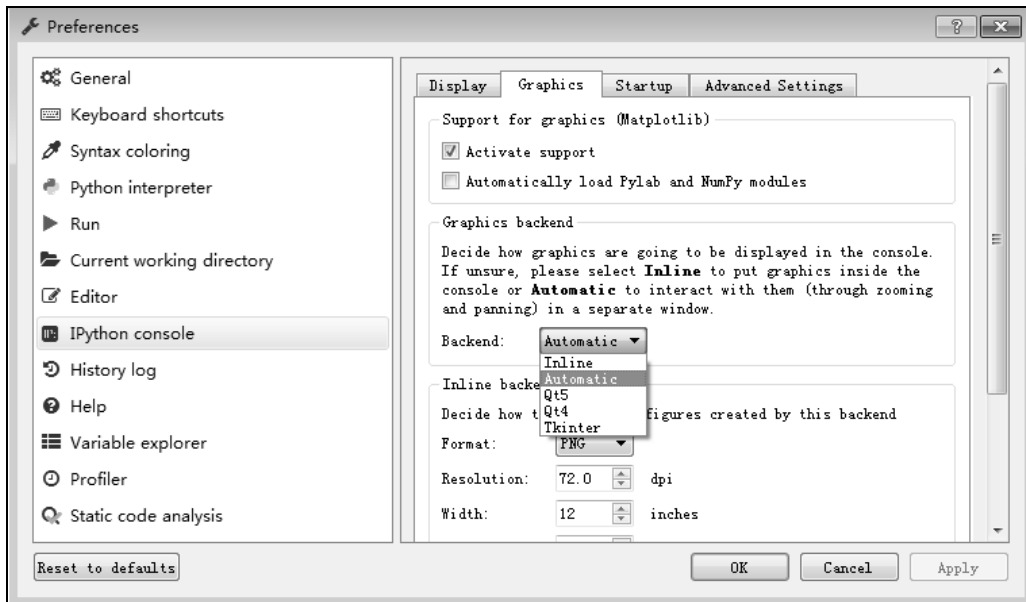


图 7-2 Spyder 软件图形设置

设置完成后重新启动 Spyder。

说明：只有在本章学习 Pyplot 时才需要选“Automatic”，后面在学习量化时要改回“Inline”。



1) Pyplot 画图的流程

(1) 绘图库加载。

```
import matplotlib.pyplot as plt
```

(2) 创建一个显示窗口 “fig1”。

```
fig1=plt.figure(' fig1')
```

我们可以创建多个窗口，如 `fig2 = plt.figure(' fig2')`。在程序中，可以使用命令 `plt.figure(fig1.number)` 选择要输出的窗口。

(3) 创建子图（不需要则可以跳过此步）：

```
plt.subplot(211)      #第一个画板的第一个子图
plt.subplot(212)      #第二个画板的第二个子图
```

(4) 设置图形标题，子图标题命名为 figure1。

```
plt.title("figure1")
```

(5) 在子图上绘图。

```
plt.plot([1, 2, 3])
```

(6) 显示 Pyplot 窗口及绘制图形。

```
plt.show()           #显示 plt 窗口及绘制图形
```

2) Pyplot 的相关说明

(1) 在绘图库加载 “`import matplotlib.pyplot as plt`” 后，用 `plt` 方法完成全部操作。

(2) 创建窗口或选择窗口用 `plt.figure()` 方法，其中创建的窗口是队列。它是先进先出的，因此，`plt.close()` 方法第一次关闭的是最早创建的一个窗口。创建的第一个窗口 “`fig1.number`” 的编号为 1，第二个窗口 “`fig2.number`” 的编号为 2。



注意

当使用 `plt.figure(2)` 时，如果没有创建编号为 2 的窗口，就会自动创建，并且当前窗口指针指向编号为 2 的窗口；如果已经存在编号为 2 的窗口，当前窗口指针就指向编号为 2 的窗口，后面的操作都是针对默认的指针指向的窗口编号进行的。

(3) 在默认窗口中, `plt.subplot()` 具有创建子图和选择子图的功能。
见示例 7-2。

```
import tushare as ts
import matplotlib.pyplot as plt
df1 = ts.get_k_data('600080', ktype='D')
fig1=plt.figure('fig1')           #创建第一个画板 (figure)
plt.subplot(211)                   #第一个画板的第一个子图
plt.title("figurea")
plt.plot(df1.open)
plt.subplot(212)                   #第二个画板的第二个子图
plt.title("figureb")
plt.plot(df1.low)
plt.legend()
fig2=plt.figure('fig2')           #创建第二个画板
plt.title("figure2")
plt.plot(df1.high)                 #默认子图命令是 subplot(111)
plt.legend()
plt.figure('fig3')                 #创建第三个画板
plt.title('figure2')               #标题
plt.plot([1,2,3])
plt.figure(fig1.number)            #选择第一个画板 (figure)
plt.subplot(211)                   #选择画板的第一个子图
plt.plot(df1.close)
plt.legend()
plt.show()                         #显示 plt 窗口及绘制图形
```

程序运行结果会出现 3 个 figure 窗口, 见图 7-3。

3. 创建绘图窗

`matplotlib.pyplot` 创建绘图窗的方法为 `figure()`, 其语法格式如下:

```
matplotlib.pyplot.figure(num=None, figsize=None, dpi=None,
facecolor=None, edgecolor=None, frameon=True, FigureClass=<class
'matplotlib.figure.Figure'>, clear=False, **kwargs)
```

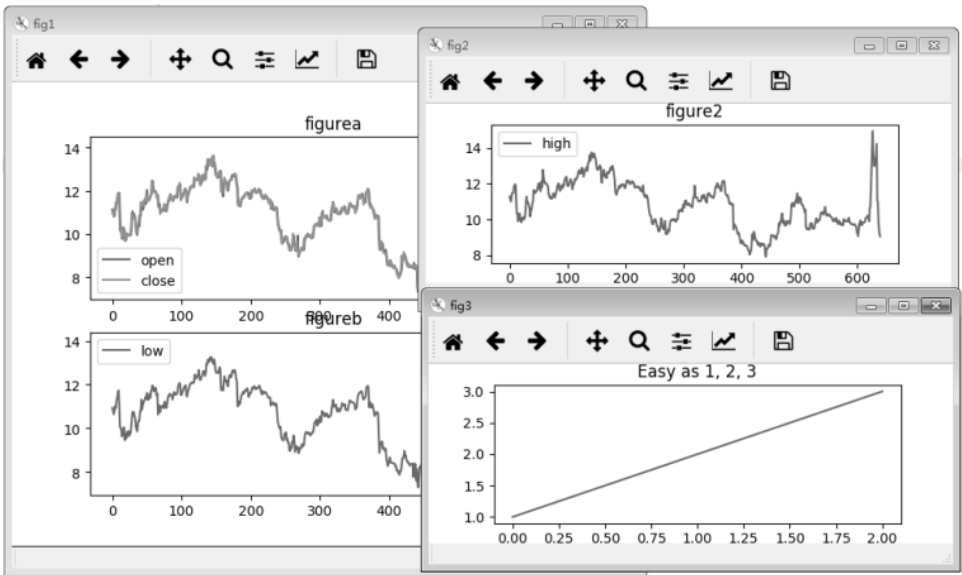



图 7-3 程序示例运行结果

matplotlib.pyplot.figure 参数及说明见表 7-3。

表 7-3 matplotlib.pyplot.figure 参数及说明

| 参数 | 说明 |
|-------------|---|
| num | 整数或字符串，可选的，默认值为 None。如果不提供 num，则会创建一个新图，使图的数量增加，图对象持有 number 属性。如果提供 num，则表明图与 id 已经存在，激活它并返回一个引用。如果这个数字不存在，则需要创建并返回它。如果是一个字符串，窗口标题则被设置为这个字符串 |
| figsize | 元组的整数,可选的,默认值为 None。宽度和高度（单位：英寸）。如果没有提供，则默认为 rcParams["figure.figsize"]=[6.4, 4.8] |
| dpi | 整数，可选的，默认值为没有。图的分辨率。如果没有提供，则默认为 rcParams["figure.dpi"]= 100 |
| facecolor | 背景颜色。如果没有提供，则默认为 rcParams["figure.facecolor"]='w' |
| edgecolor | 边框颜色。如果没有提供，则默认为 rcParams["figure.edgecolor"]='w' |
| frameon | Bool，可选的，默认值为 True。如果为 False，则抑制图纸图框 |
| FigureClass | Figure 的子类。选择使用一个自定义 Figure 的实例 |
| clear | Bool，可选的，默认值为 Flase。如果为 True，则说明 Figure 已经存在，那么把它清除 |

返回值：Figure 返回一个图像的窗口，它允许定制 Figure 到 Pyplot 绘图类。此外，kwargs 将传递到 Figure 的 init()函数。

下面见示例 7-3。

```
import time
import matplotlib
import matplotlib.pyplot as plt
print('matplotlib.__version__ = ',matplotlib.__version__)
f1=plt.figure('abc', (4,4),100,facecolor='#F0F0F0')
plt.title("figure1")
f2=plt.figure('hello',None,None,'#FFD700','#FF0000')
plt.title("figure2")
plt.show()
time.sleep(10)      #等待 10 秒
plt.close()         #关闭一个窗口
time.sleep(5)       #等待 5 秒
plt.close()         #关闭一个窗口
```

4. 创建一个子绘图区域

创建一个子绘图区域“subplot”的格式：

```
matplotlib.pyplot.subplot(*args, **kwargs)
```

参数说明：

*** args:** 一个三位数的整数或三个整数描述次要情节的位置。如果三个整数分别是“nrows”网格行、“ncols”网格列与子图位置号，则索引从 1 开始在左上角增加。

****kwargs:** 这种方法可以设置子图属性，参数非常多，其中常用的参数有 **anchor**, **facecolor**, **fc**, **label**, **title**, **xlabel**, **visible**, **ylabel** 等。

例如，利用“**fc='r'**”更改背景色；利用“**title='子图标题'**”修改标题。

通常用法如下：

```
subplot(nrows, ncols, index, **kwargs)
subplot(pos, **kwargs)
subplot(ax)
```

如果没有用 **figure()** 方法创建窗口，那么在第一次使用 **subplot()** 方法时，就会自动创建 **Figure1** 窗口。

ax1=plt.subplot(221) 和 **ax1=plt.subplot(2,2,1)** 的功能相同，但是 **subplot(221)** 只能是 111~999 的数字，不能为 0，且每段不能超过 10。**subplot(2,2,1)** 可以超过 10，如

subplot(12,14,4)等。例如，subplot(111)表示把图形分为1行1列，且激活第1个子图区域绘图。子图区域从1开始数，先行后列。subplot(3,4,5)表示图形分为3行4列，且激活第5个子图区域绘图。

例如，创建一个子绘图区域“ax1”。

```
ax1=plt.subplot(224, sharex=ax1, fc='r',title='bbb')
```

删除子绘图区域：

```
plt.delaxes(ax1)
```

#删除子图画画

见示例 7-4。

```
import matplotlib
import matplotlib.pyplot as plt
print('matplotlib.__version__ = ',matplotlib.__version__)
f1=plt.figure('abc', (4,4),100,facecolor='#F0F0F0') #创建绘图窗口
plt.title("figure1")
ax1=plt.subplot(221) #增加子图
plt.title("figure1")
ax2=plt.subplot(2, 2, 4) #增加子图
plt.title("figure2")
ax3=plt.subplot(222, frameon=False) #增加子图
plt.title("figure3")
plt.plot([1,2,3])
ax4=plt.subplot(223, projection='polar') #增加子图
plt.delaxes(ax1) #删除子图画画
plt.subplot(ax1) #显示子图画
```

程序运行结果见图 7-4。

5. 创建一个网格子绘图区域

创建一个网格子绘图区域“subplot2grid()”的语法格式如下：

```
matplotlib.pyplot.subplot2grid(shape, loc, rowspan=1, colspan=1,
fig=None, **kwargs)
```

subplot2grid()参数及说明见表 7-4。

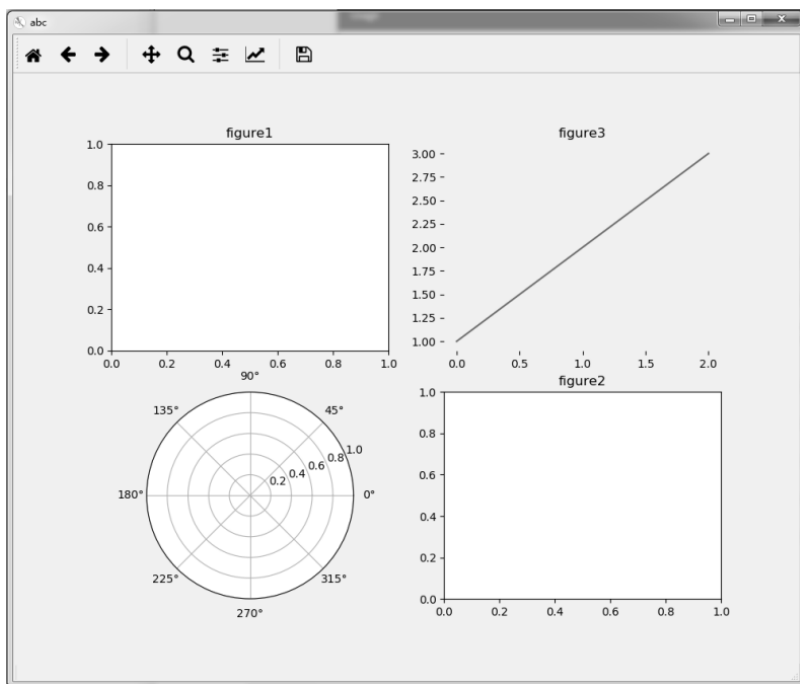


图 7-4 程序示例运行结果

表 7-4 subplot2grid()参数及说明

| 参数 | 说明 |
|----------------|-----------------------------------|
| shape | 形状，整型列表数据。网格的形状轴的地方，第一个为行数，第二个为列数 |
| loc | 网格内轴位置，整型列表数据。第一个为行数，第二个为列数 |
| rowspan | 行宽，整型类型，为 axis 跨越的行数 |
| colspan | 列宽，整型类型，为 axis 跨越的列数 |
| fig | Figure 窗口，可选，默认为当前的图 |
| facecolor 或 fc | 颜色 |
| **kwargs | 关键字参数 |

通常用法如下：

```
ax=subplot2grid(shape, loc, rowspan=1, colspan=1)
```

如果没有用 figure()方法创建窗口，那么在第一次使用 subplot2grid()方法时，就会自动创建 Figure1 窗口。

见示例 7-5。

```
import matplotlib.pyplot as plt
ax1 = plt.subplot2grid((3,3), (0,0), colspan=3)
plt.plot([1,2,3])
ax2 = plt.subplot2grid((3,3), (1,0), colspan=2)
plt.plot([3,5,3])
ax3 = plt.subplot2grid((3,3), (1, 2), rowspan=2)
plt.plot([5,3,1])
ax4 = plt.subplot2grid((3,3), (2, 0))
plt.plot([1,5,2])
ax5 = plt.subplot2grid((3,3), (2, 1))
plt.suptitle("subplot2grid")
plt.plot([5,1,5])
```

程序运行结果见图 7-5。

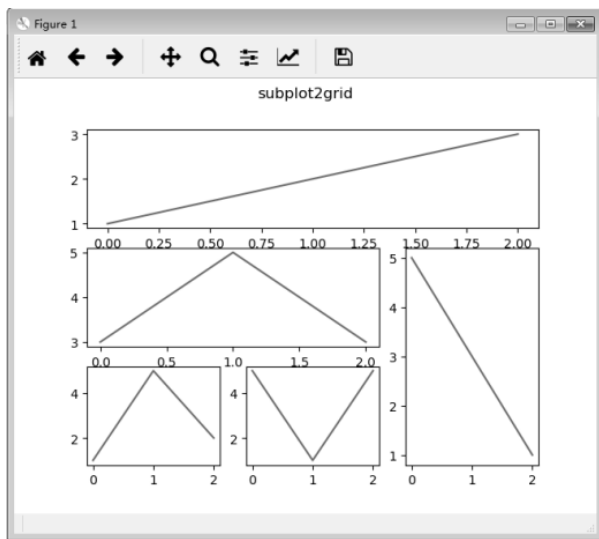


图 7-5 程序示例运行结果

6. 调整子绘图区域布局

为了让绘图区更美观,我们可以使用 `subplots_adjust()` 命令调整子图之间的布局。命令格式如下:

```
matplotlib.pyplot.subplots_adjust(left=None, bottom=None, right=None,
```

```
top=None, wspace=None, hspace=None)
```

pyplot.subplots_adjust 参数说明和默认值见表 7-5。

表 7-5 pyplot.subplots_adjust 参数说明和默认值

| 参数 | 说明 | 默认值/% |
|--------|--------|-------|
| left | 左边距 | 0.125 |
| right | 右边距 | 0.9 |
| bottom | 底部边距 | 0.1 |
| top | 顶部边距 | 0.9 |
| wspace | 子图间列宽度 | 0.2 |
| hspace | 子图间行宽度 | 0.2 |

我们可以通过改变参数观察布局的变化，见示例 7-6。

```
#调整子绘图区域布局 subplots_adjust
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
np.random.seed(20190902)
#编造区间[0,1]中的一些数据
y = np.random.normal(loc=0.6, scale=0.5, size=1000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))
plt.figure('figure 窗口')
plt.title('画 linear')
#画 linear
plt.subplot(221)
plt.plot(x, y)
plt.yscale('linear')
plt.title('画 linear')
plt.grid(True)
#画 log
plt.subplot(222)
plt.plot(x, y)
```

```

plt.yscale('log')
plt.title('画 log')
plt.grid(True)
#画 symmetric log
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthreshy=0.01)
plt.title('画 symlog')
plt.grid(True)
#画 logit
plt.subplot(224)
plt.plot(x, y)
plt.yscale('logit')
plt.title('画 logit')
plt.grid(True)
plt.gca().yaxis.set_minor_formatter(NullFormatter())
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95,
hspace=0.25, wspace=0.35)
plt.show()

```

程序运行结果见图 7-6。

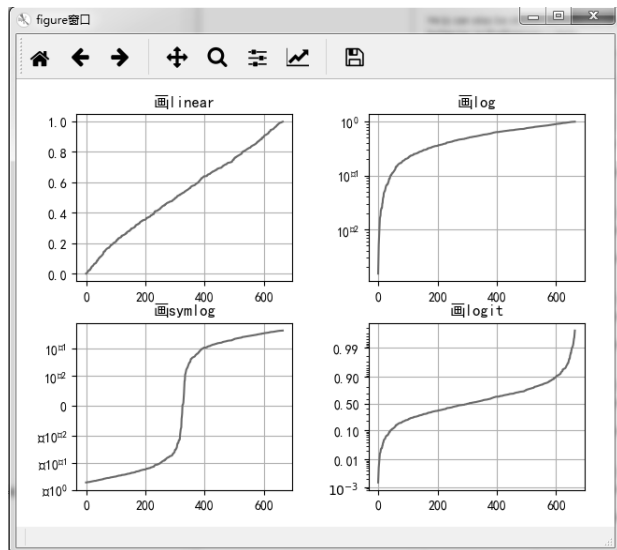


图 7-6 程序示例运行结果

7. 创建一个轴图

Axes 层也称轴图、坐标层，是建立在 Figure 上的一个坐标系。

命令格式如下：

```
matplotlib.pyplot.axes(arg=None, **kwargs)
```

一般用法：

```
plt.axes()
plt.axes(rect, projection=None, polar=False, **kwargs)
plt.axes(ax)
```

axes()方法的参数说明见表 7-6。

表 7-6 axes()方法的参数说明

| 参数 | 说明 |
|-----------------------------------|--|
| arg=None | 使用 subplot(111, **kwargs) 添加一个新的窗口轴 |
| arg=[left, bottom, width, height] | 添加一个新的轴图，尺寸矩形规范化 (0,1) 单位 |
| projection | 投影，参数为 {None, 'aitoff', 'hammer', 'lambert', 'mollweide', 'polar', 'rectilinear', str}，可选的投影类型为 Axes，默认为 None，结果在一个直线的投影 |
| sharex | 共享 x 轴 |
| sharey | 共享 y 轴 |
| Axes | Axes（或一个子类 Axes），返回的轴类取决于所使用的投影 |
| **kwargs | 关键字参数 |

下面给出示例 7-7。

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']          #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False           #用来正常显示负号
np.random.seed(20190902)
#创建一些数据
dt = 0.001
t = np.arange(0.0, 10.0, dt)
r = np.exp(-t[:1000] / 0.05)                        #impulse response
x = np.random.randn(len(t))
s = np.convolve(x, r)[:len(x)] * dt                 #colored noise
```



```

#主要 axes 默认 subplot(111)
plt.plot(t, s)
plt.axis([0, 1, 1.1 * np.min(s), 3 * np.max(s)])
plt.xlabel('time (s)')
plt.ylabel('current (nA)')
plt.title('高斯彩色噪声')
#这是一个嵌入在主 axes 的子 axes
a = plt.axes([.65, .6, .2, .2], facecolor='w')
n, bins, patches = plt.hist(s, 400, density=True)
plt.title('概率')
plt.xticks([])
plt.yticks([])

plt.show()

```

程序运行结果见图 7-7。

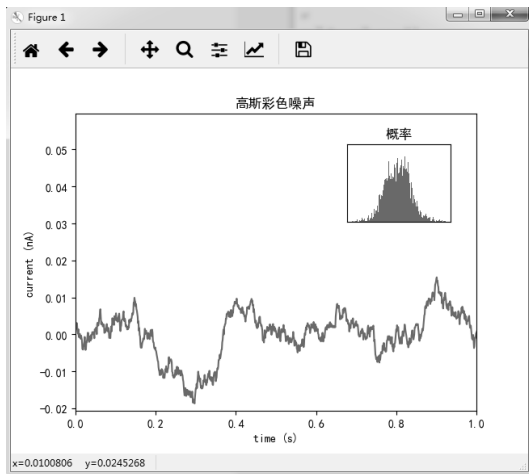


图 7-7 程序示例运行结果

7.2 有关Pyplot显示的方法

本节集中把有关显示方面的方法介绍给大家。

为了能够正确地显示中文，我们需要在程序的开始增加两行命令进行字体和字符集的设置。

```
plt.rcParams['font.sans-serif']=['SimHei']      #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False       #用来正常显示负号
```

Pyplot 的方法命令格式如下：

plt.方法名

Pyplot 的方法及功能和调用格式，见表 7-7。

表 7-7 Pyplot 的方法

| 方法 | 功能 | 调用格式 |
|--------------|------------------------|--|
| figure() | 创建一个显示窗口 | plt.figure(num=1,figsize=(8,8)) |
| cla() | 清空当前 axes 区 | plt.cla() |
| clf() | 清空当前 figure 窗口 | plt.clf() |
| close([fig]) | 关闭 figure 窗口 | plt.close() |
| draw() | 重画当前 figure 窗口 | plt.draw() |
| gcf() | 获得当前 figure 窗口大小 | fig=plt.gcf() |
| grid() | 显示当前子图坐标网格，默认为 True | plt.grid(True) |
| legend() | 允许在绘图区域中放置图注 | plt.legend(['A simple line']) |
| sca(ax) | 激活 ax 子图 | plt.sca(ax1) |
| show | 显示窗口 | plt.draw() |
| suptitle() | 给当前 figure 窗口增加标题 | plt.suptitle(' figure title', fontsize=12) |
| title | 设置子图标题（与 subplot 结合使用） | plt.title(' subplots 1') |
| xlabel | x 坐标标签 | plt.xlabel('周期') |
| ylabel | y 坐标标签 | plt.ylabel('价格') |

见示例 7-8。

```
import pandas as pd
import matplotlib.pyplot as plt
import tushare as ts
import time
ds='2017-06-01'                                #开始日期
de=time.strftime('%Y-%m-%d',time.localtime(time.time())) #今天的日期
plt.rcParams['font.sans-serif']=['SimHei']      #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False       #用来正常显示负号
df1 = ts.get_k_data('600030',ktype='D',start=ds,end=de)
```

```

ma5=pd.Series.rolling(df1.close, 5).mean() #股票收盘价 5 日均线
ma20=pd.Series.rolling(df1.close, 20).mean() #股票收盘价 20 日均线
fig=plt.figure() #建立新窗口
ax=plt.subplot(111) #设置子图
plt.plot(df1.close) #显示股票 close
plt.plot(ma5) #显示 MA5
plt.plot(ma20) #显示 MA20
plt.legend(['CLOSE', 'MA5', 'MA20']) #设置标签
plt.grid() #显示网格
plt.suptitle('股票价格平均线', fontsize=12) #设置图形标题
plt.xlabel('周期') #显示 x 轴标签
plt.ylabel('价格') #显示 y 轴标签
plt.show() #显示 plt 窗口

```

程序运行结果见图 7-8。

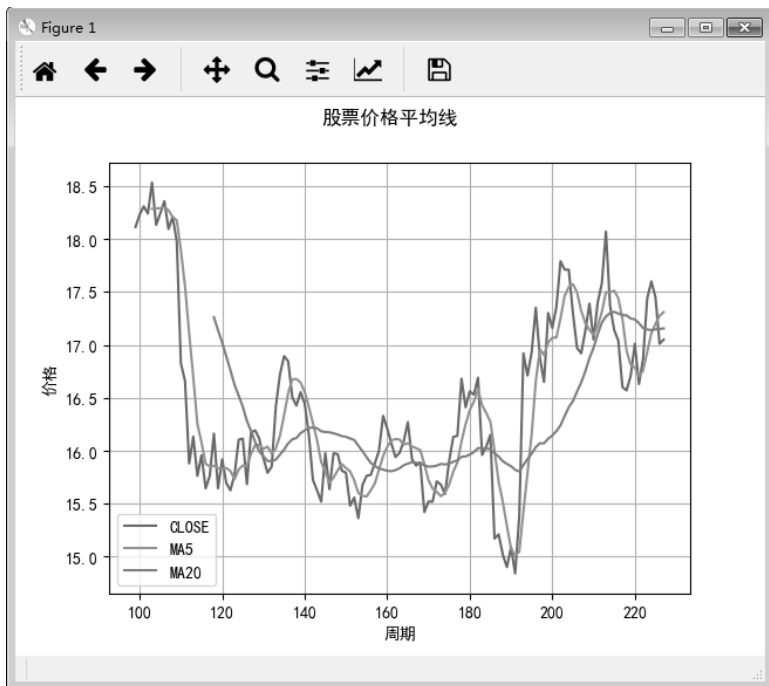


图 7-8 程序示例运行结果

7.3 Pyplot常用绘图方法

matplotlib.pyplot 的绘图方法很多，其中常用的如下：

- (1) plt.plot(x,y,label,color,width): 根据 x,y 数组绘制直线、曲线。
 - (2) plt.boxplot(data,notch,position): 绘制一个箱型图。
 - (3) plt.bar(left,height,width,bottom): 绘制一个条形图。
 - (4) plt.barh(bottom,width,height,left): 绘制一个横向条形图。
 - (5) plt.polar(theta,r): 绘制极坐标图。
 - (6) plt.pie(data,explode): 绘制饼图。
 - (7) plt.psd(x, NFFT=256, pad_to, Fs): 绘制功率谱密度图。
 - (8) plt.specgram(x, NFFT=256, pad_to, F): 绘制谱图。
 - (9) plt.cohere(x,y,NFFT=256,Fs): 绘制“x-y”的相关性函数。
 - (10) plt.scatter(): 绘制散点图，其中 x,y 是长度相同的序列。
 - (11) plt.step(x,y,where): 绘制步阶图。
 - (12) plt.hist(x,bins,normed): 绘制直方图。
 - (13) plt.contour(X,Y,Z,N): 绘制等值线。
 - (14) plt.clines(): 绘制垂直线。
 - (15) plt.stem(x,y,linefmt, markerfmt, basefmt): 绘制曲线每个点到水平轴线的垂线。
 - (16) plt.plot_date(): 绘制数据日期。
- 其中，plt.plot()方法是画线程序，命令格式如下：

```
matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None,
**kwargs)
```

这个方法用法很多，其中简单的用法有 plot([x], y, [fmt], data=None, **kwargs)。其中，x, y 是函数值 $y=f(x)$ ，fmt 是线的属性。data 是关于(x,y)的数据对象。**kwargs 用于指定 Line2D 的属性，如线、抗锯齿、颜色等。

字符串格式如下：

```
fmt='[color][marker][line]'
```

其中，color 支持颜色缩写，见表 7-8。

表 7-8 color

| 字符 | 颜色 |
|-----|-----|
| 'b' | 蓝色 |
| 'g' | 绿色 |
| 'r' | 红色 |
| 'c' | 青色 |
| 'm' | 品红色 |
| 'y' | 黄色 |
| 'k' | 黑色 |
| 'w' | 白色 |

marker 的描述见表 7-9。

表 7-9 marker

| 字符 | 描述 |
|-----|-------------------|
| '.' | 点标记 |
| ',' | 像素标记 |
| 'o' | 圆圈标记 |
| 'v' | triangle_down 标记 |
| '^' | triangle_up 标记 |
| '<' | triangle_left 标记 |
| '>' | triangle_right 标记 |
| '1' | tri_down 标记 |
| '2' | tri_up 标记 |
| '3' | tri_left 标记 |
| '4' | tri_right 标记 |
| 's' | 方块 |
| 'p' | 实心五角 |
| '*' | 星标记 |
| 'h' | hexagon1 标记 |
| 'H' | hexagon2 标记 |
| '+' | + 标记 |
| 'x' | x 标记 |
| 'D' | 钻石标记 |
| 'd' | thin_diamond 标记 |
| ' ' | vline 标记 |
| '_' | 线标记 |

line 的描述见表 7-10。

表 7-10 line

| 字符 | 描述 |
|------|---------------|
| '_' | 实线风格 |
| '--' | 虚线样式 |
| '-.' | dash-dot 线条样式 |
| '.' | 点线 |

见示例 7-9，划分为两个图，上面显示收盘价 5 日和 20 日均线，下面显示成交量和成交量 5 日均线。

```
import pandas as pd
import matplotlib.pyplot as plt
import tushare as ts
import time

ds='2019-06-01' #开始日期
de=time.strftime('%Y-%m-%d',time.localtime(time.time())) #今天的日期
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
df1 = ts.get_k_data('600030',ktype='D',start=ds,end=de)
ma5=pd.Series.rolling(df1.close, 5).mean() #股票收盘价 5 日均线
ma20=pd.Series.rolling(df1.close, 20).mean() #股票收盘价 20 日均线
plt.figure(2, figsize=(12,8), dpi=80)
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
ax1 = plt.subplot(211)
plt.plot(df1.close,'.-') #显示股票 close
plt.plot(ma5,'k:') #显示 MA5
plt.plot(ma20,'b-') #显示 MA20
plt.legend(['CLOSE','MA5','MA20']) #设置标签
plt.grid() #显示网格
plt.suptitle('股票价格平均线', fontsize=12) #设置图形标题
plt.xlabel('周期') #显示 x 轴标签
plt.ylabel('价格') #显示 y 轴标签
ax2 = plt.subplot(212)
plt.bar(df1.index,df1.volume)
```

```

vma5=pd.Series.rolling(df1.volume, 5).mean() #股票收盘价 5 日均线
plt.plot(vma5,'b' ) #显示 VMA5
plt.legend(['volume','VMA5']) #设置标签
plt.ylabel('成交量') #显示 y 轴标签
plt.show() #显示 plt 窗口

```

程序运行结果见图 7-9。

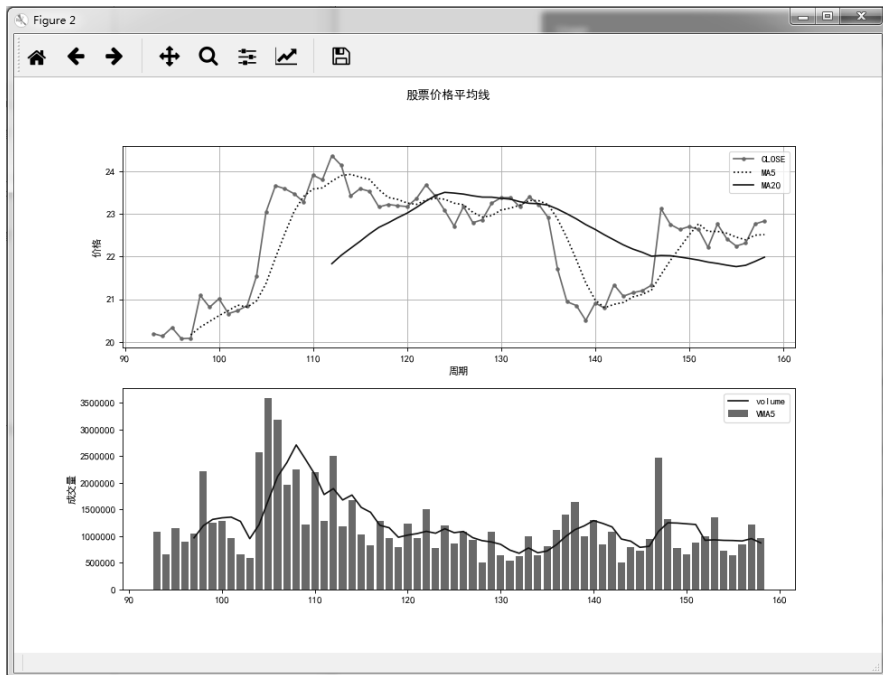


图 7-9 示例程序运行结果

7.4 共享x坐标轴画图

假如我们想把两个子图合成一个,即共享 x 坐标轴画图,这时就需要使用 `twinx()` 方法。

命令格式:

```
matplotlib.pyplot.twinx(ax=None)
```

例如, `ax2` 和 `ax1` 轴图共享 x 轴。

```
ax2 = plt.twinx(ax1) #ax2 和 ax1 轴图共享 x 轴
```

如果把成交量和收盘价走势图放到一起，那么图形就一定非常乱，而我们关注的重点是收盘价的趋势，成交量仅仅是判断趋势的一个因素。为了不使成交量喧宾夺主，可以把成交量和成交量均线都设置为透明的，这样就主次分明了。其中，参数 `alpha` 代表透明度，1 是不透明，0 是透明，我们设置为 `alpha=.2`。

见示例 7-10 的代码。

```
import pandas as pd
import matplotlib.pyplot as plt
import tushare as ts
import time
ds='2019-06-01' #开始日期
de=time.strftime('%Y-%m-%d',time.localtime(time.time())) #今天的日期
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
df1 = ts.get_k_data('600030',ktype='D',start=ds,end=de)
ma5=pd.Series.rolling(df1.close, 5).mean() #股票收盘价 5 日均线
ma20=pd.Series.rolling(df1.close, 20).mean() #股票收盘价 20 日均线
plt.figure(2, figsize=(12,8), dpi=80)
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
ax1 = plt.subplot(111)
plt.plot(df1.close,'.-') #显示股票 close
plt.plot(ma5, ) #显示 MA5
plt.plot(ma20,) #显示 MA20
plt.legend(['CLOSE','MA5','MA20']) #设置标签
plt.grid() #显示网格
plt.suptitle('股票价格平均线', fontsize=12) #设置图形标题
plt.xlabel('周期') #显示 x 轴标签
plt.ylabel('价格') #显示 y 轴标签
ax2 = plt.twinx(ax1)
#ax3 = plt.subplot2grid((7,4), (4,0),sharex=ax2,rowspan=1, colspan=4)
plt.bar(df1.index,df1.volume,facecolor='#386d13', alpha=.2)
vma5=pd.Series.rolling(df1.volume, 5).mean() #股票收盘价 5 日均线
plt.plot(vma5,'m' , alpha=.2) #显示 VMA5
plt.legend(['volume','VMA5']) #设置标签
plt.ylabel('成交量') #显示 y 轴标签
plt.show() #显示 plt 窗口
```


程序运行结果见图 7-10。

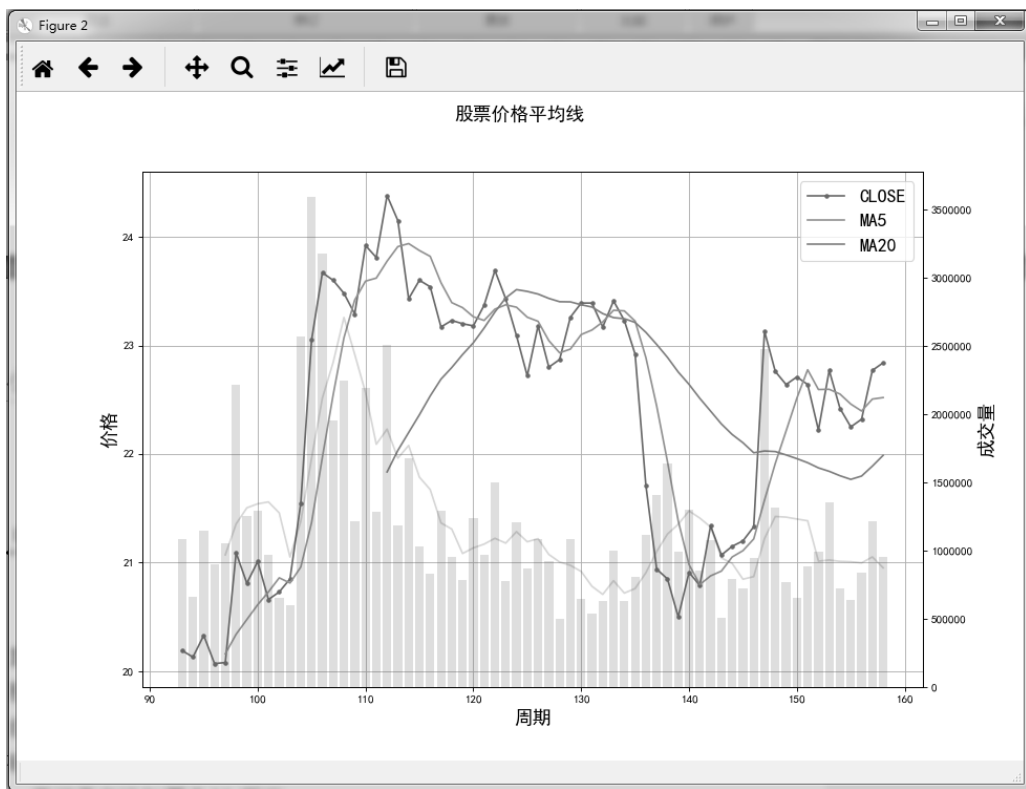


图 7-10 示例程序运行结果

7.5 绘制K线图

K 线图是股票软件中最常见的图形。K 线又称为蜡烛图，起源于日本。K 线图有直观、立体感强、携带信息量大的特点，蕴涵着丰富的东方哲学思想，能充分显示股价趋势的强弱、买卖双方力量平衡的变化，且能较准确地预测后市走向。目前，已经形成了一套完善的 K 线分析理论。技术分析中的重要流派——K 线派，就是专门以研究 K 线的形状和组合为基础的。

其记录方法如图 7-11 所示：

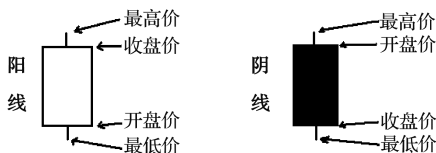


图 7-11 K 线画法

日 K 线是根据股价（指数）在一天的走势中形成的四个价位——开盘价、收盘价、最高价、最低价绘制而成的。

在收盘价高于开盘价时，开盘价在下收盘价在上，二者之间的长方柱用红色或空心绘出，称为阳线。其上影线的最高点为最高价，下影线的最低点为最低价。

在收盘价低于开盘价时，开盘价在上收盘价在下，二者之间的长方柱用黑色或实心绘出，称为阴线。其上影线的最高点为最高价，下影线的最低点为最低价。

股票行情数据主要有“date”日期时间，“close”收盘价，“high”最高价，“low”最低价，“open”开盘价，“volume”成交量。在股票软件公式系统中（不区分大小写字母，即 C 同 c），行情数据除了用全称外，还可以使用简称，如 c 代表 close，h 代表 high，l 代表 low，o 代表 open，v 代表 volume。

当获取的股票行情数据的列表名都不相同时，先要转化为自己定义的列表格式“[index, date, open, close, high, low, volume]”。

我们使用系统默认的画 K 线的 `candlestick_ohlc()` 方法。

(1) 在 matplotlib 2.0.2 版本中自带此方法，使用如下语句。

```
from matplotlib.finance import candlestick_ohlc #仅用于matplotlib 2.0.2 版
```

(2) 在 matplotlib 3.0.2 版中，如果取消了此方法，就需要自行安装 `mpl_finance` 包。

```
pip install mpl_finance
```

安装成功后，使用如下语句。

```
from mpl_finance import candlestick_ohlc #仅用于matplotlib 3.0.2 版
```

使用 `candlestick_ohlc()` 方法绘制出来的 K 线不是连续的，在周末和节假日 K 线就会留下很大一段空白，这非常影响对 K 线走势的观察。因此，我们要做一些修改以去掉图形中的空白。

见示例 7-11。

```
#显示K线图和5日, 20日均线
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import ticker as mticker
import tushare as ts
import time
#from matplotlib.finance import candlestick_ohlc #matplotlib2.0.0 用
from mpl_finance import candlestick_ohlc #matplotlib3.0.0 用
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
ds='2019-06-01' #开始日期
de=time.strftime('%Y-%m-%d',time.localtime(time.time())) #今天的日期
df1 = ts.get_k_data('600030',ktype='D',start=ds,end=de)
df2=df1.copy()
df2.dropna(inplace=True) #删除无效数据
date_tickers=df2['date'] #刻度值
del df2['date']
df2['date']=df2.index
ma5=pd.Series.rolling(df2.close, 5).mean() #股票收盘价5日均线
ma20=pd.Series.rolling(df2.close, 20).mean() #股票收盘价20日均线
fig, ax = plt.subplots()
days = df2.reindex(columns=['date','open','high','low','close'])
ax.tick_params(axis='y', colors='b')
ax.tick_params(axis='x', colors='b')
plt.title('股票K线图')
plt.ylabel('Stock price')
candlestick_ohlc(ax, days.values, width=.6, colorup='#ff1717',
colordown='#53c156')
ax.plot(days.date.values,ma5,label='MA5', linewidth=1.5)
ax.plot(days.date.values,ma20,label='MA20', linewidth=1.5)
ax.grid(True, color='r')
ax.xaxis.set_major_locator(mticker.MaxNLocator(6)) #x轴分成几等分
ax.xaxis.label.set_color("b")
ax.yaxis.label.set_color("b")
plt.legend(['MA5','MA20']) #显示图中右上角的提示信息
plt.show()
```

程序运行结果见图 7-12。

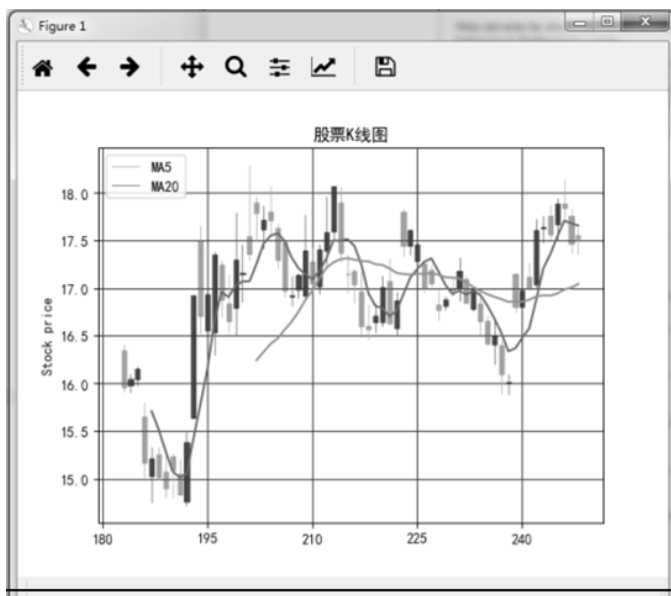


图 7-12 示例程序运行结果

有关 Matplotlib 的更多功能，感兴趣的读者可以去官网继续学习。

另外，如果你使用的是 Spyder 软件，别忘记了把图形输出改回去哦！即在 Spyder 软件中，“Tools > Preferences > IPython console > Graphics > Backend”选“Inline”。

8

第 8 章

Tkinter 模块

Tkinter 是 Python 的标准 GUI 库，它是一个跨平台的脚本图形界面接口，能够开发各种图形界面且可以快速创建 GUI 应用程序。由于 Tkinter 已经内置到 Python 的安装包中，因此只要安装好 Python 就能导入 Tkinter 库。

Tkinter 的前身是 Tcl/Tk 的 Python GUI 工具组，其中 Tcl/Tk 是由 C 语言编写的 Python 图形扩展库，其实现了类似于 Windows 视窗中 X 视窗系统的功能。

由于小白量化投资软件采用的就是 Tkinter 搭建的图形框架，因此读者可以用 Tkinter 开发各种应用程序窗口和应用界面。

在 Tkinter 中，图形单元称为部件（Widget），也称为控件或组件。

8.1 Tkinter 的使用

在新版本中，Tkinter 库是内置库，因此不需要另外安装。

Tkinter 模块在 Python 2 中的库名为 Tkinter，在 Python 3 中的库名为 tkinter。

在 Python 2 中，装载库命令如下：

```
import Tkinter
```

#仅用于 Python 2

在 Python 3 中，装载库命令如下：

```
import tkinter
```

#仅用于 Python 3

为了便于书写程序和区分不同库中的函数或方法，我们一般会给 Tkinter 模块加



上“tk”别名。

```
import tkinter as tk
```

#仅用于 Python 3

本书主要讲解 Python 3 中 Tkinter 的使用，并且使用别名“tk”。

1. Tkinter 的使用流程

(1) 导入 Tkinter 模块。

```
import tkinter as tk
```

#导入 Tkinter3 模块

(2) 创建主窗口。

```
root =tk.Tk()
```

#创建 Tkinter 窗口对象

(3) 创建控件，如创建 Label 标签控件等。

```
label=tk.Label(root,text='窗口内容在此') #创建 Label 控件
```

(4) 把控件通过 pack 布局管理方法放到窗口中。

```
label.pack()
```

#pack() 方法用来设置控件在窗口中的位置、大小等

(5) 进入 Tkinter 主循环。

```
root.mainloop()
```

#进入 Tkinter 消息循环

见示例 8-1。

```
import tkinter as tk
```

#导入 Tkinter3 模块

```
root =tk.Tk()
```

#创建 Tkinter 窗口对象

```
root.title(string = '窗口标题')
```

#设置窗口标题

```
background = tk.Label(root,text = '窗口内容在此') #创建 Label 控件
```

```
background.pack()
```

#pack() 方法用来设置控件在窗口中的位置、大小等

```
root.mainloop()
```

#进入 Tkinter 消息循环

程序运行结果见图 8-1。

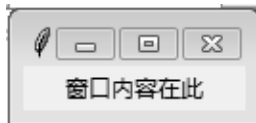


图 8-1 示例程序运行结果

Tkinter 创建的第一个 Tk() 对象是 Tkinter 的主窗口对象。Tk 主窗口主要是管理 Tkinter 的应用程序，因此一般称为 root，有些程序员喜欢用 window 表示，还有些程序员喜欢用 app 表示。

Tkinter 有很多不同的控件，如 Button, Canvas, Frame, Label 等。在有些容器控件中还可以放置其他的控件，如 Frame, LabelFrame, Canvas, Text, PanedWindow, Notebook 等；普通控件不能放置其他控件，如 Button, Label 等。

控件在容器中放置的几何管理方法有顺序放置“pack()”方法、网格放置“grid()”方法、绝对位置放置“place()”方法。

Tkinter 程序设计的总体思路：一般来说，设计窗口或控制面板最好使用 Frame.grid() 方式，即在 Frame 中填充不同颜色进行调整，调整完成后再把所需要的控件用 pack() 方法放置到 Frame 容器中。

2. Tkinter 的变量类

Tkinter 有自己的变量类，这与 Python 变量有点区别。Tkinter 的变量类主要用于设置 Tkinter 控件，或绑定 Tkinter 控件。如果 Tkinter 的变量值改变了，Tkinter 系统就可以使用跟踪功能更新那些设置了这个变量的相关控件中关键字的属性值，即通过修改 Tkinter 变量值来动态实现改变 Tkinter 控件的属性。

例如，Label 控件的 textvariable 关键字属性就需要设置一个 Tkinter 变量 StringVar。如果 StringVar 变量值被修改，Label 文本就会自动更新 StringVar 变量值。

在 Tkinter 系统中，有变量类的基类“Variable”，基类又派生出其他变量类，如 StringVar（字符型）类、IntVar（整型）类、DoubleVar（双精度浮点型）类和 BooleanVar（布尔型）类。

这些由基类派生出来的变量类除变量数值类型不同外，其他属性和方法都是相同的，因此下面我们只介绍 StringVar 类的用法。

在使用过程中，先要创建 Tkinter 变量对象，其语法格式如下：

```
var=tk.Variable(master=None, value=None)
```

其中，如果 master 省略或值为 None，则自动设置为 Tkinter 全局变量“Tk() 对象 root”。在创建 Tkinter 变量对象的语句之前，必须先创建 Tk() 全局对象 root，即在 root=tk.TK() 语句后才能正常创建 Tkinter 变量对象。

value 可实现创建变量对象的同时赋值给 value 参数值。

Tkinter 变量对象的主要常用操作有两个：设置值 “.set()” 方法和获取值 “.get()” 方法。具体用法如下：

(1) 设置 Tkinter 变量对象的值。

```
root =tk.Tk()                #创建 Tkinter 窗口对象
var=tk.StringVar ()          #创建 StringVar() 字符变量对象
var.set('新值')              #设置变量
```

(2) 获取 Tkinter 变量对象的值。

```
var=tk.StringVar ()          #创建 StringVar() 字符变量对象
var.set('新值')              #设置变量 var
str=var.get()                 #获取 var 变量对象中的值
```

除了上面两种方法外，还包括一些跟踪（trace）类的方法，这些类都以 “trace” 字符串开头。具体用法如下：

(1) 添加一个变量的跟踪，trace_variable 方法同 trace 方法。

```
cbname = var.trace(mode, callback)
cbname = var.trace_variable(mode, callback)
```

添加一个变量的跟踪，用于在某种 “mode” 被触发的时候调用 callback 函数。mode 模式字符串有 r, w, u 三种模式，其中 r 表示跟踪的变量被读取，w 表示跟踪的变量被改写，u 表示跟踪的变量被删除或未赋值。

(2) 删除一个变量的跟踪。

```
var.trace_vdelete(mode, cbname)
```

cbname 是 trace_variable() 方法在创建跟踪时返回的字符串。

(3) 返回所有跟踪回调信息。

```
var.trace_vinfo()
```

返回所有变量跟踪的 callback 信息。

下面看一个关于 StringVar 对象的演示，示例 8-2。

```
import tkinter as tk          #装载 tkinter 模块，用于 Python 3
root=tk.Tk()                  #创建 Tkinter 主窗口
root.title("Tkinter 变量演示")
root.attributes('-topmost',1) #参数 1，设置顶层窗口，覆盖其他窗口
```



```

var=tk.StringVar(root,value='这是 Tkinter 变量测试 ! \n')
root.update()
def callbackw(*args):                                #修改操作
    x=args
    print(x)
    print('\n 变量有修改操作: '+var.get())
def callbackw2(*args):                                #修改操作 2
    print('\n 变量有修改操作 2: '+var.get())
def callbackr(*args):                                #读取操作
    x=args
    print(x)
    print('\n 变量有读取操作: '+var.get())
cbw=var.trace('w', callbackw)                          #创建变量跟踪
cbw2=var.trace('w', callbackw2) #创建变量跟踪
cbr=var.trace('r', callbackr)                          #创建变量跟踪
var.set(var.get()+'修改变量值可改变 Lable 标签! ')    #修改 Tk 变量
label=tk.Label(root,textvariable=var).pack()           #显示标签
print(var.trace_vinfo())                               #打印全部跟踪信息
print('删除 cbw2')
var.trace_vdelete('w', cbw2)                           #删除变量跟踪
print(var.trace_vinfo())                               #打印全部跟踪信息
print('删除全部跟踪')
cbw2=var.trace('w', callbackw2)                         #创建变量跟踪
print(var.trace_vinfo())                               #打印全部跟踪信息
root.mainloop()

```

程序运行会产生一个窗口，窗口中放置了一个标签，标签内容来自 StringVar 对象变量 var，并且对 var 的修改和获取进行跟踪。

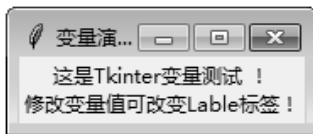


图 8-2 程序运行结果

下面是程序运行期间的输出跟踪信息：

```
('PY_VAR0', '', 'r')
```

```
变量有读取操作 1: 这是 Tkinter 变量测试 !
变量有修改操作 2: 这是 Tkinter 变量测试 !
修改变量值可改变 Lable 标签!

('PY_VAR0', '', 'w')
变量有修改操作: 这是 Tkinter 变量测试 !
修改变量值可改变 Lable 标签!

('PY_VAR0', '', 'r')
变量有读取操作: 这是 Tkinter 变量测试 !
修改变量值可改变 Lable 标签!

[('r', '146335560callbackr'), ('w', '183655496callbackw2'), ('w',
'146313160callbackw')]
删除 cbw2

[('r', '146335560callbackr'), ('w', '146313160callbackw')]
删除全部跟踪

[]
```

8.2 Tkinter控件的属性

由于 Tkinter 控件都继承于 Widget 类, 因此它们拥有共同的属性和方法。

8.2.1 Tkinter 控件的标准属性

下面介绍一些控件的标准属性, 见表 8-1。个别属性是一些控件的特有属性, 如 command 属性只有按钮 Button 控件才会有。

表 8-1 Tkinter 控件的属性

| 属性关键字 | 说明及属性取值 |
|------------------|----------------------------|
| activebackground | ①设置在活动状态时的背景色。 ②默认值系统指定 |
| activeforeground | ①设置在活动状态时的前景色。 ②默认值系统指定 |

续表

| 属性关键字 | 说明及属性取值 |
|---------------------|---|
| anchor | ①控制文本或者图像在 label 中的显示位置。 ②值为 N, NE, E, SE, S, SW, W, NW, CENTER。(E, W, S, N 分别表示东、西、南、北) ③默认值为 CENTER |
| bg 或 background | 设置背景色 |
| bitmap | ①指定显示到 Label 上的位图。 ②如果指定了 image, 则该选项忽略。 ③Python 内置了 10 种位图, 如 error, gray75, gray50, gray25, gray12, hourglass, info, questhead, question, warning, 它们可以直接使用, 设置 bitmap 即可 |
| bd 或 bordwidth | ①指定边框宽度。 ②默认值由系统指定, 通常为 1 像素或 2 像素 |
| compound | ①文本和图像混合模式。 ②在默认情况下, 如果指定位图和图片, 则不显示文本。 ③如果选项设置为 CENTER, 则文本显示在图像上。 ④如果设置为 BOTTOM, LEFT, RIGHT, TOP, 那么图像则显示在文本的旁边。假如为 BOTTOM, 图像则在文本的下方。 ⑤默认值是 None |
| command | 按钮消息的回调函数。当按钮被点击时, 执行该函数 |
| cursor | ①指定当前鼠标在 Label 上飘过时的样式。 ②默认系统指定 |
| disbledforeground | ①指定 Label 在不可用时的前景色。 ②默认系统指定 |
| fg 或 foreground | 设置前景色 |
| Font | ①指定字体样式和大小。 ②默认由系统指定。 ③使用元组设置, 如'微软雅黑',10 |
| focus_set | 当前部件得到的焦点 |
| height | ①设置 Label 的高度。 ②如果 Label 是文本, 则单位是文字高度, 否则为像素。 ③如果 Label 为 0 或者默认, 则自动根据 Label 计算 |
| highlightbackground | ①指定当 Label 没有获得焦点时的高亮边框颜色。 ②默认系统指定 |
| highlightcolor | ①当 Label 获得焦点时的高亮边框颜色。 ②默认系统指定 |

续表

| 属性关键字 | 说明及属性取值 |
|--------------------|--|
| highlightthickness | ①指定高亮边框的宽度。 ②默认值是 0 |
| Image | ①指定图像。 ②该值应该是 PhotoImage, BitmapImage, 或者能兼容的对象。 ③该选项优先于 text 和 bitmap, 属性关键字 |
| justify | 在显示多行文本时, 设置不同行之间的对齐方式。 ①定义对齐方式。 ②取值为 LEFT, RIGHT, CENTER。 ③文本位置取决于属性关键字“anchor”的选项。 ④默认值为 CENTER |
| Padx | x 轴间距, 单位为像素, 在 x 轴方向上的内边距 |
| Pady | y 轴间距, 单位为像素, 在 y 轴方向上的内边距 |
| relief | 设置控件 3D 效果。 ①指定边框样式。 ②默认值是 FLAT。 ③可取值为 SUNKEN, FLAT, RIDGE, SOLID, GROOVE |
| State | ①指定 Label 的状态。 ②这个标签控制 Label 如何显示。 ③默认值是 NORMAL。 ④可设置为 ACTIVE 和 DISABLED |
| takefocus | ①如果值是 True (真), 则该 Label 接收输入焦点。 ②默认值为 False (假) |
| Text | ①指定文本。 ②文本可包含换行符。 ③如果设置属性关键字 image 或 bitmap, 则该选项可被忽略 |
| textvariable | 指定一个变量名, 变量值被转换为字符串在控件上显示。当变量值改变时, 控件也将自动更新。 ①Label 显示 Tkinter 变量, StringVar 类型。 ②如果变量被修改, Label 文本就自动更新 |
| underline | 默认按钮上的文本都不带下画线。取值就是带下画线的字符串索引, 当值为 0 时, 第一个字符带下画线; 当值为 1 时, 前两个字符带下画线, 以此类推 |
| Width | ①设置 Label 的宽度。 ②如果 Label 显示文本, 则单位为文本长度, 否则为像素。 ③设置为 0 或者默认就自动计算 |
| wraplength | 限制按钮每行显示的字符数量。 ①决定 Label 文本将被分成多少行。 ②该选项指定每行的长度, 单位是屏幕单元。 ③默认值为 0 |



修改控件的关键字属性值有三种方式。

(1) 在创建对象时，使用关键字参数。例如：

```
label = tk.Label(root, text='label', fg = 'red', bg = 'blue')
```

其中，关键字参数有 text, fg, bg。在创建对象后，将选项名称视为字典索引。

(2) 更改或配置控件属性。

由于控件的属性类似于字典索引，因此可以用修改字典数据的方式修改控件属性。例如：

```
label ["fg"] = "red"
label ["bg"] = "green"
```

(3) 使用 config() 方法更新控件对象创建后的多个属性值。

例如，同时修改 Label 的多个属性。

```
label.config(fg = 'yellow', bg = 'blue')
```

下面看一个修改控件属性的示例 8-3。

```
import time
import tkinter as tk                #装载 tkinter 模块，用于 Python 3
root=tk.Tk()                        #创建 Tkinter 主窗口
labels=[]                           #标签列表
for color in ['blue','black','red']:
    f = tk.Frame(root, borderwidth=1,bg='black')
    label=tk.Label(f, text=color, width=10,fg=color,bg='yellow',
highlightcolor='white')            #用关键字参数设置属性
    label.pack(side=tk.LEFT)
    labels.append(label)            #把新建立的 Label 对象保存到列表中
    f.pack(side=tk.LEFT)
root.update()                       #刷新 Tkinter 窗口
time.sleep(5)                       #暂停 5 秒钟
#部件的字典方式修改属性
labels[0]['bg']='green'
labels[0]['fg']='yellow'
labels[1]['bg']='blue'
labels[1]['fg']='yellow'
```

```
#使用 config() 方法修改多个属性
labels[2].config(fg = 'yellow', bg = 'red')
root.mainloop()                                #进入消息循环
```

程序运行结果见图 8-3。



(前图是修改前, 后图是修改后)

图 8-3 程序运行结果

8.2.2 Tkinter 控件属性的值和单位

由于 Tkinter 控件的不同属性具有不同的功能, 因此它们的取值和单位都不相同。下面我们介绍控件的常用取值和单位。

1. 维度单位

Tkinter 的维度单位是各种长度“Lengths”、宽度“Widths”和其他控件维度单位的描述。

(1) 如果一个维度被设置为一个整数, 那么它就是像素单位。

(2) 可以指定一个包含数字和维度单位“c, i, m, p”的字符串, 其中 c 表示厘米, i 表示英寸, m 表示毫米, p 表示打印机的点。

Tkinter 控件的大小通常用宽度“width”和高度“height”表示, 单位为屏幕像素。另外, x 轴间距“padx”和 y 轴间距“pady”, 以及描述边框宽度的高亮边框宽度“highlightthickness”和边框宽度“borderwidth”的单位也是像素。

2. 坐标系统

Tkinter 坐标系统的基本单位是像素, 左上角像素坐标 (0,0) 称为原点, 坐标值为整数。在 Tkinter 图形显示系统中, 原点在左上角, x 坐标增加向右边移动, y 坐标增加向底部移动, 见图 8-4。

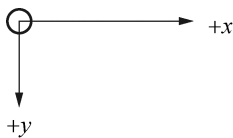


图 8-4 Tkinter 坐标系

3. 颜色

Tkinter 的颜色用字符串表示，有两种表示方法。

(1) Tkinter 系统规定了一些颜色字符串，如 white, black, red, green, blue, cyan, yellow, magenta 等。

(2) 可以用十六进制的字符串指定不同颜色，如红 (R)、绿 (G)、蓝 (B) 三种颜色不同比例的混合色。Tkinter 自定义颜色常用字符串格式表示，其中白色用 “#FFFFFF” 表示、红色用 “#RR0000” 表示、金色用 “#FFD700” 表示。

Tkinter 控件的颜色属性是常用属性，属性关键字有 activebackground, activeforeground, background, foreground, highlightbackground, highlightcolor 等。

十六进制的字符串 #00 至 #FF 表示十进制的数字 0 至 255。我们可以用下列方法把数字转换为颜色字符串。

```
newcolor = '#%02x%02x%02x'%(0,255,0)    #设置你最喜欢的 rgb 颜色
```

为了方便使用 Tkinter 颜色，我们定义了下面两个函数。

```
def tkcolor(r,g,b):
    #把 r, g, b 数字颜色转换为 Tkinter 自定义颜色
    return '#%02x%02x%02x'%(r,g,b)

def RGB(tkcolor):
    #把 Tkinter 自定义颜色分解为数字 r, g, b
    r=eval('0x'+tkcolor[1:3])
    g=eval('0x'+tkcolor[3:5])
    b=eval('0x'+tkcolor[5:7])
    return r,g,b
```

4. 字体类型

我们会根据 Python 使用的操作系统平台来指定 Tkinter 字体类型的风格，设置字体类型的属性关键字为 `font`。

字体类型使用元组表示，其中第一个元素是字体，其后是字体大小，之后的字符串包含一个或多个风格修饰符。其中，`bold` 表示加粗，`normal` 表示正常字体宽度，`italic` 表示斜体，`roman` 表示正体。

例如，`("Helvetica", "16")` 和 `("Times", "24", "bold")`。

```
label=tk.Label(root, text='字体大小',font = ("Times", "24",
"bold"),bg='#1E488F')
```

5. 对齐方式

Tkinter 控件的对齐方式的属性关键字为 `anchor`。`anchor` 取值有 N（上）、NE（右上）、E（右）、SE（右下）、S（下）、SW（左下）、W（左）、NW（左上）、CENTER（中间）等。对齐方式见图 8-5。

| | | |
|--------|------------|--------|
| NW(左上) | N(上) | NE(右上) |
| W(左) | CENTER(中间) | E(右) |
| SW(左下) | S(下) | SE(右下) |

图 8-5 Anchor 属性取值含义

在程序中使用对齐方式属性时，要加上别名“tk”，如 `tk.W`。

6. 浮雕风格

Tkinter 控件的浮雕风格的属性关键字为 `relief`，能够设置的取值有 `RAISED`，`SUNKEN`，`FLAT`，`RIDGE`，`GROOVE`，`SOLID` 等，在使用时也要加上别名“tk”。

下面我们通过示例程序来看看运行的浮雕风格效果，见示例 8-4。

```
#Tkinter 边框效果 relief
import tkinter as tk                                #导入 Tkinter3 模块
```



```

root =tk.Tk()                                #创建 Tkinter 窗口对象
row = [None]*5
for bdw in range(5):
    row[bdw] = tk.Frame(root, borderwidth=0)
    tk.Label(row[bdw], text='borderwidth = %d ' % bdw).pack(side=tk.LEFT)
    i = 0
    column = []
    for relief in [tk.RAISED,tk.SUNKEN, tk.FLAT,tk.RIDGE, tk.GROOVE,
                  tk.SOLID]:
        column.append(tk.Frame(row[bdw], borderwidth=bdw, relief=relief))
        tk.Label(column[i], text=relief, width=10).pack(side=tk.LEFT)
        column[i].pack(side=tk.LEFT, padx=7-bdw, pady=5+bdw)
        i += 1
    row[bdw].pack()
root.mainloop()

```

程序运行结果见图 8-6。

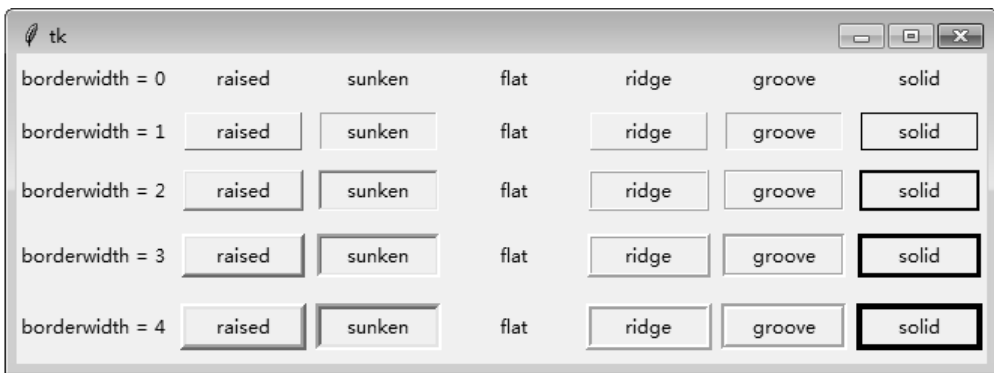


图 8-6 程序运行结果

7. 位图

在 Tkinter 控件中，位图选项的关键字为 `bitmap`，位图选项关键字的值见图 8-7。



图 8-7 Tkinter 控件的位图选项

在图 8-7 中, 从左到右的字符串分别表示 error, gray75, gray50, gray25, gray12, hourglass, info, questhead, question, warning。

8. 鼠标样式

Tkinter 控件的鼠标样式是指鼠标在控件上飘过时的样式, 属性关键字为 `cursor`。Windows 系统提供的鼠标样式见图 8-8。














































































| | | | |
|---|--|---|--|
|  arrow |  man |  circle |  rtl_logo |
|  based_arrow_down |  middlebutton |  clock |  sailboat |
|  based_arrow_up |  mouse |  coffee_mug |  sb_down_arrow |
|  boat |  pencil |  cross |  sb_h_double_arrow |
|  bogosity |  pirate |  cross_reverse |  sb_left_arrow |
|  bottom_left_corner |  plus |  crosshair |  sb_right_arrow |
|  bottom_right_corner |  question_arrow |  diamond_cross |  sb_up_arrow |
|  bottom_side |  right_ptr |  dot |  sb_v_double_arrow |
|  bottom_tee |  right_side |  dotbox |  shuttle |
|  box_spiral |  right_tee |  double_arrow |  sizing |
|  center_ptr |  rightbutton |  draft_large |  spider |
|  draft_small |  spraycan |  left_ptr |  umbrella |
|  draped_box |  star |  left_side |  ur_angle |
|  exchange |  target |  left_tee |  watch |
|  fleur |  tcross |  leftbutton |  xterm |
|  gobbler |  top_left_arrow |  ll_angle |  X_cursor |
|  gumby |  top_left_corner |  lr_angle | |
|  hand1 |  top_right_corner | | |
|  hand2 |  top_side | | |
|  heart |  top_tee | | |
|  icon |  trek | | |
|  iron_cross |  ul_angle | | |

图 8-8 鼠标样式

如果把部件属性 `cursor` 设置为字符串 `hand2`, 当把鼠标移动到部件时, 鼠标形状就变为手的图形。如果设置部件属性 `cursor='heart'`, 则鼠标形状就变为心的形状。

9. 图片

在 Tkinter 应用程序中, 图片属性关键字为 `image`。有三种获取图像文件的方法, 它们支持不同的图像文件格式。

1) 显示位图（双色）图像

显示位图（双色）图像使用.xbm 文件格式，需要使用位图构造函数 `BitmapImage`，其语法格式如下。

```
bmp=tk.BitmapImage ( file=f[, background=b][, foreground=c] )
```

其中，参数 `file` 的值 `f` 是.xbm 图像文件的名称。图像像素占用 1bit（1 个二进制像素位）。在通常情况下，图像中的前景用 1 表示，显示为黑色像素；背景用 0 表示，显示为透明的背景。如果要改变这种显示背景的行为，就可以使用可选参数“`background = b`”选项设置背景颜色 `b`，可选参数“`foreground=c`”设置前景颜色 `c`。

例如，如果要想显示一个图像标签就使用一个标签控件，其中参数 `image` 使用 `BitmapImage` 对象图像的 `logo`，相关代码如下。

```
logo = tk.BitmapImage("logo.xbm", foreground='red')
tk.Label( image=logo ).grid()
```

2) 显示彩色图像

彩色图像的文件有.gif, .pgm 和.ppm 文件格式，显示彩色图像需要构造函数 `PhotoImage`，其语法格式如下。

```
img=tk.PhotoImage ( file=f )
```

其中，`file` 的值 `f` 是图像文件名，其可以返回在任何地方都能使用的 Tkinter 图像数据 `img`。

3) 使用 Python 成像库

Python 成像库“PIL”支持更广泛格式的图像，它是专门为在 Tkinter 应用程序中显示图像而设计的模块。示例 8-5。

```
import tkinter as tk                #导入 Tkinter
from PIL import Image, ImageTk
root = tk.Tk()
root.title(string = '图片演示')     #设置窗口标题
root.geometry('500x300')
photo=Image.open("abc.jpg")         #读入数据
img=ImageTk.PhotoImage(photo)       #转换为 Tkinter 图像格式
label_img = tk.Label(root, image = img)
label_img.pack()
root.mainloop()
```

图像 img 可以在 Tkinter 任何使用图像 image 的地方使用。Tk.PhotoImage() 函数可以使用更多种类的图像格式文件，如常用的.jpg 图片格式文件。

8.3 Tkinter 主窗口

在使用 Tkinter 开发程序时，为了更好地管理 Tkinter 程序，首先需要使用 Tk() 方法建立主窗口。

一般只有执行 mainloop() 主循环的方法才能运行 Tkinter 命令的动作和创建 Tkinter 应用窗口。但是在主循环启动前，可以使用 update() 方法执行 Tkinter 程序的一些操作命令，如创建窗口、建立控件并放置在窗口中，但是执行完毕后 Tkinter 程序就会停止。只有 mainloop() 方法才会启动 Tkinter 消息事件循环操作，这时才会接收 Tkinter 的动作，如事件处理、鼠标或键盘操作处理、点击按钮操作动作等。

下面介绍 Tk 主窗口的主要操作。

1. 获取屏幕和窗口信息

我们可以用下面方法获取屏幕和窗口信息，单位是像素，窗口大小是指像素多少。

| | |
|--|----------------------|
| import tkinter as tk | # 导入 Tkinter3 模块 |
| root = tk.Tk() | # 创建 Tkinter 窗口对象 |
| screenwidth = root.winfo_screenwidth() | # 获取屏幕宽度 (单位: 像素) |
| screenheight = root.winfo_screenheight() | # 获取屏幕高度 (单位: 像素) |
| win_width = root.winfo_width() | # 获取窗口宽度 (单位: 像素) |
| win_height = root.winfo_height() | # 获取窗口高度 (单位: 像素) |
| x = root.winfo_x() | # 获取窗口 x 坐标 (单位: 像素) |
| y = root.winfo_y() | # 获取窗口 y 坐标 (单位: 像素) |

2. 设置窗口

设置窗口主要是改变窗口的标题、大小、位置、状态等操作。

1) 设置窗口标题

Tkinter 主窗口对象 “root” 包含很多窗口操作的方法，如 Wm 类方法和 Tkinter 中定义的操作方法，其中 Tkinter 中定义的操作方法不包含字符 “wm_”。

| | |
|-----------------------|---------------------|
| root.wm_title('窗口标题') | # Wm 类设置窗口标题方法 |
| root.title('窗口标题') | # Tkinter 中设置窗口标题方法 |

2) 设置窗口长宽和窗口的位置

Tkinter 为我们提供了改变窗口大小和放置位置的操作方法。`root.geometry()`方法可以根据输入的屏幕宽高、坐标来改变 Tkinter 窗口的位置和大小。`root.geometry()`方法需要以字符串的方式提供参数,格式如 `root.geometry('300x200+150+250')`,其中参数含义为屏幕宽度为 300 像素、高度为 200 像素、x 坐标为 150、y 坐标为 250。由于参数中没有变量或其他字符,因此我们可以用字符串格式命令 “`'{ }x{ }+{ }+{ }'.format(width,height, x, y)`” 输入整型变量参数 “`width,height, x, y`”。

```
width=300      #把窗口宽度(单位:像素)300赋值给变量width
height=200     #把窗口高度(单位:像素)300赋值给变量height
x,y=150,250    #给屏幕坐标(x,y)赋值(100,200)
root.geometry('{ }x{ }+{ }+{ }'.format(width,height, x, y))
               #改变窗口位置和大小
```

如果仅仅想改变窗口的大小,如宽度为 600 像素、高度为 500 像素,就可以用格式 “`root.geometry('600x500')`”,或者用字符串格式 “`'{ }x{ }'.format(width,height)`”。

```
width=300      #把窗口宽度(单位:像素)300赋值给变量width
height=200     #把窗口高度(单位:像素)300赋值给变量height
root.geometry('{ }x{ }'.format(width,height)) #改变窗口大小
```

3) 设置窗口的新图标

设置窗口的新图标可以使用 Tkinter 的 `root.iconbitmap(bitmap)`方法,参数 `bitmap` 是包含路径的 .ico 文件名。

```
root.iconbitmap('tt.ico')
```

4) 窗口的其他操作

Tkinter 还提供很多其他有关窗口的操作方法,其中比较常用的如下。

(1) 设置最顶层窗口。

Tkinter 窗口的 `root.attributes('-topmost',1)`方法可以把窗口设置为顶层窗口,即覆盖其他窗口,总是在最顶层显示,这个属性主要针对 Windows 系统平台。

```
root.attributes('-topmost',1)    #参数1,设置顶层窗口,覆盖其他窗口
root.attributes('-topmost',0)    #参数0,正常窗口,允许其他窗口覆盖
```

(2) 设置工具栏样式窗口。

Tkinter 窗口的 `root.attributes("-toolwindow", 1)`方法可以把窗口设置为工具栏样

式窗口，即窗口顶栏没有了最大化和最小化的按钮，只有关闭窗口按钮。

```
root.attributes("-toolwindow", 1)    #参数 1，设置工具栏样式窗口
                                     #参数 1，没有最大化和最小化的按钮
root.attributes("-toolwindow", 0)    #参数 0，设置为普通窗口
```

(3) 隐藏窗口标题栏。

Tkinter 窗口的 `root.overrideredirect(True)` 方法可以隐藏窗口标题栏，即不显示窗口标题和最大化、最小化、关闭的按钮。而 `root.overrideredirect(False)` 方法可恢复显示窗口标题栏。

在使用中，如果隐藏了关闭按钮，也可以用 Windows 热键 “Alt+F4” 关闭窗口。

```
root.overrideredirect(True)          #参数 True，隐藏窗口标题栏
root.overrideredirect(False)         #参数 False，恢复显示窗口标题栏
```

(4) 窗口的最大化和最小化。

Tkinter 窗口的 `root.state("zoomed")` 方法可以使窗口最大化。

`root.state("iconic")` 方法可以使窗口隐藏，只在 Windows 系统程序栏显示程序图标，相当于使窗口最小化。

`root.state("normal")` 方法可以使窗口还原为普通状态。

说明：Tkinter 窗口的 `root.state()` 方法仅用于 Windows 系统。

```
root.state("zoomed")                 #窗口最大化
root.state("iconic")                 #隐藏窗口，相当于窗口最小化
root.state("normal")                 #设置为普通窗口
```

(5) 设置全屏窗口。

Tkinter 窗口的 `root.attributes("-fullscreen", True)` 方法可以把窗口设置为全屏窗口。在全屏状态下，可以用 Windows 热键 “Alt+F4” 关闭窗口。

`root.attributes("-fullscreen", False)` 方法可以取消全屏窗口，还原为普通窗口。

```
root.attributes("-fullscreen", True)  #设置全屏窗口
root.attributes("-fullscreen", False) #取消全屏窗口，还原为普通窗口
```

(6) 设置窗口透明度。

Tkinter 窗口的 `root.attributes("-alpha", n)` 方法可以设置窗口的透明度。透明度参数 `n` 是从 0 到 1 的小数，1 是不透明，0 是全透明。

```
root.attributes("-alpha", 0.8)        #设置窗口的透明度为 0.8
```

8.4 Toplevel 顶层子窗口

Tkinter 运行后出现一个主窗口，或称根窗口。如果要继续创建一些子窗口，就可以使用 `tk.Toplevel()` 方法。

`Toplevel` 子窗口与 `Tk` 根窗口相比，除了 `Toplevel` 子窗口无 Tkinter 系统控制权外，拥有的属性和方法基本都是一样的，如修改窗口标题、建立菜单、改变窗口大小、移动位置、放置控件等操作。

示例 8-6，演示创建一个子窗口。

| | |
|---|-------------------------------|
| <code>import tkinter as tk</code> | <code>#导入 Tkinter3 模块</code> |
| <code>root =tk.Tk()</code> | <code>#创建 Tkinter 窗口对象</code> |
| <code>root.title(string = '主窗口')</code> | <code>#设置窗口标题</code> |
| <code>root.geometry('300x200')</code> | <code>#改变窗口大小</code> |
| <code>top=tk.Toplevel(root)</code> | <code>#创建 Toplevel 窗口</code> |
| <code>top.title(string = 'top 子窗口')</code> | <code>#设置窗口标题</code> |
| <code>top.geometry('200x100')</code> | <code>#改变窗口大小</code> |
| <code>top.attributes("-toolwindow", 1)</code> | <code>#参数 1，设置工具栏样式窗口</code> |
| <code>root.mainloop()</code> | <code>#进入 Tkinter 消息循环</code> |

程序运行结果见图 8-9。



图 8-9 示例程序运行结果

从上例中可以看到，根窗口的普通方法也可以用于 `top` 子窗口中。

8.5 创建窗口菜单条

Tkinter 的菜单 Menu 能用于根窗口和子窗口中。

菜单使用过程如下：

(1) 创建主菜单条。

```
import tkinter as tk          #导入 Tkinter3 模块
root =tk.Tk()                 #创建 Tkinter 窗口对象
mainmenu = tk.Menu(root)      #创建主菜单
```

(2) 建立子菜单。

```
filemenu = tk.Menu(mainmenu, tearoff=False) #建立子菜单 filemenu,
tearoff=False 不显示分割虚线
helpmenu = tk.Menu(mainmenu, tearoff=True) #建立子菜单 helpmenu
```

(3) 把子菜单加入到主菜单中。

```
mainmenu.add_cascade(label="菜单",menu=filemenu)
mainmenu.add_cascade(label="帮助", menu=helpmenu)
```

(4) 建立子菜单项目。

```
filemenu.add_command(label="读取", command=hello) #增加 filemenu 的菜单项目
filemenu.add_command(label="保存",command=hello)
filemenu.add_separator()                        #增加 filemenu 的分割线
filemenu.add_command(label="另存",command=hello)
filemenu.add_command(label="查看",command=hello)
```

(5) 给窗口配置菜单。

```
root.config(menu=mainmenu)          #给根窗口配置菜单条
```

主菜单演示见示例 8-7。

```
'''
tkinter 演示。
菜单演示
'''
```




```

import tkinter as tk                #导入 Tkinter3 模块
def hello():
    print( "hello!")
root =tk.Tk()                        #创建 Tkinter 窗口对象
root.title(string = '主窗口标题')   #设置窗口标题
root.geometry('300x200')            #改变窗口大小
top=tk.Toplevel(root)               #创建 Toplevel 窗口
top.title(string = 'top 子窗口')     #设置窗口标题
top.geometry('200x100')              #改变窗口大小
# (1) 创建主菜单条。
mainmenu = tk.Menu(root,relief=tk.RIDGE,cursor='CLOCK')  #创建主菜单
# (2) 建立子菜单。
filemenu = tk.Menu(mainmenu, tearoff=False,cursor='CLOCK')
#建立子菜单 filemenu, tearoff=False 不显示分割虚线
helpmenu = tk.Menu(mainmenu, tearoff=True,selectcolor='red' )
#建立子菜单 helpmenu
# (3) 把子菜单加入到主菜单中。
mainmenu.add_cascade(label="菜单",menu=filemenu)
mainmenu.add_cascade(label="帮助", menu=helpmenu)
# (4) 建立子菜单项目。
filemenu.add_command(label="读取", command=hello) #增加 filemenu 的菜单项目
filemenu.add_command(label="保存",command=hello)
filemenu.add_separator()                      #增加 filemenu 的分割线
filemenu.add_command(label="另存",command=hello)
filemenu.add_command(label="查看",command=hello)
#-----
helpmenu.add_command(label="关于",command=hello) #增加 helpmenu 的菜单项目
helpmenu.add_separator()                      #增加 filemenu 的分割线
helpmenu.add_command(label="使用方法",command=hello)
# (5) 给根窗口配置菜单。
root.config(menu=mainmenu)                  #给根窗口配置菜单
top.config(menu=mainmenu)                   #给子窗口配置菜单
root.mainloop()                             #进入消息循环

```

程序运行结果见图 8-10。



图 8-10 示例程序运行结果

8.6 创建弹出菜单

Tkinter 的弹出菜单能用于很多 Thinter 控件上。弹出菜单和窗口主菜单的创建过程相似，只是配置菜单的方式不同。

弹出菜单的使用过程：

(1) 创建弹出主菜单条。

```
import tkinter as tk          #导入 Tkinter3 模块
root =tk.Tk()                 #创建 Tkinter 窗口对象
menu = tk.Menu(root, tearoff=True) #创建弹出菜单
```

(2) 建立子菜单。

如果有多级弹出菜单，就需要创建弹出子菜单。

```
filemenu = tk.Menu(menu, tearoff=False) #建立弹出子菜单 filemenu
```

(3) 建立子菜单项目，把弹出二级子菜单放到所需的位置。

```
menu.add_command(label="Undo", command=hello)
menu.add_command(label="Redo", command=hello)
menu.add_command(label="Copy", command=hello)
menu.add_command(label="Paste", command=hello)
menu.add_command(label="Cut", command=hello)
```

```

menu.add_separator()                #增加 filemenu 的分割线
menu.add_cascade(label="菜单", menu=filemenu)
menu.add_command(label="SelectAll", command=hello)

filemenu.add_command(label="读取", command=hello) #增加 filemenu 的菜单项目
filemenu.add_command(label="保存", command=hello)
filemenu.add_separator()            #增加 filemenu 的分割线
filemenu.add_command(label="另存", command=hello)
filemenu.add_command(label="查看", command=hello)

```

(4) 创建弹出菜单操作函数。

```

def popup(event):                    #弹出菜单操作函数
    menu.post(event.x_root, event.y_root)

```

(5) 把弹出菜单绑定到所需的控件上。

```

frame.bind("<Button-3>", popup)      #把弹出菜单绑定到所需的控件上

```

弹出菜单能够绑定到很多 Tkinter 控件上, 特别是 Text, TreeView 等, 合理使用弹出菜单能够提高用户对软件操作的快捷性。

下面见示例 8-8。

```

'''
tkinter 演示。
菜单演示
'''
import tkinter as tk                #导入 Tkinter3 模块
#移动窗口到屏幕中央
def setCenter(root,w,h):
    ws = root.winfo_screenwidth()
    hs = root.winfo_screenheight()
    x = int( (ws/2) - (w/2) )
    y = int( (hs/2) - (h/2) )
    root.geometry('{}x{}+{}+{}'.format(w, h, x, y))
def hello():
    print( "hello!")
counter = 0

```

```
def update():
    global counter
    counter = counter + 1
    menu.entryconfig(0, label=str(counter)+" 运行次数")
root =tk.Tk()                #创建 Tkinter 窗口对象
root.title(string = '主窗口标题')#设置窗口标题
background = tk.Label(root,text = '窗口内容在此')#创建 Label
background.pack()            #pack() 方法用来设置窗口位置, 大小等选项
setCenter(root,400,300)      #把主菜单移动的屏幕中央,菜单窗口大小为 400 像素×
                              300 像素

#建立一个弹出菜单
menu = tk.Menu(root, tearoff=True)#创建弹出菜单
filemenu = tk.Menu(menu, tearoff=False)
#建立弹出子菜单 filemenu, tearoff=False 不显示分割虚线

menu.add_command(label="Undo", command=hello)
menu.add_command(label="Redo", command=hello)
menu.add_command(label="Copy", command=hello)
menu.add_command(label="Paste", command=hello)
menu.add_command(label="Cut", command=hello)
menu.add_separator()         #增加 filemenu 的分割线
menu.add_cascade(label="菜单",menu=filemenu)
menu.add_command(label="SelectAll", command=hello)
filemenu.add_command(label="读取", command=hello)#增加 filemenu 的菜单项目
filemenu.add_command(label="保存",command=hello)
filemenu.add_separator()     #增加 filemenu 的分割线
filemenu.add_command(label="另存",command=hello)
filemenu.add_command(label="查看",command=hello)
#create a canvas
frame = tk.Frame(root, width=512, height=512)
frame.pack()
def popup(event):             #弹出菜单操作函数
    menu.post(event.x_root, event.y_root)
frame.bind("<Button-3>", popup)#把弹出菜单绑定到所需的控件上
root.mainloop()              #进入消息循环
```

程序运行结果见图 8-11。



图 8-11 示例程序运行结果

8.7 控件的几何布局管理方法

Tkinter 控件共有三种几何布局管理器，分别是 Pack 布局、Grid 布局和 Place 布局，通过几何布局管理器可以控制窗体或容器中各个控件的位置关系。

1. Pack 布局

在使用 `pack()` 方法时，默认先使用的放到上面，然后依次向下排，其会给控件一个自认为合适的位置和大小，这是默认方式。`pack()` 方法不能和 `grid()` 方法混用。

使用 `pack()` 方法显示一个控件 `w`，其语法格式如下。

```
w.pack(option=value, ...)
```

其中，`option` 可以使用一些关键字。`pack()` 方法属性的关键字及说明见表 8-2。

表 8-2 `pack()` 方法属性关键字

| 关键字 | 说明 |
|--------|---|
| after | 将控件置于其他控件之后 |
| before | 将控件置于其他控件之前 |
| anchor | 控件的对齐方式：顶对齐“N”，底对齐“S”，左对齐“W”，右对齐“E”。 值为 N, NE, E, SE, S, SW, W, NW, CENTER |
| side | 控件在主窗口中的位置，可以为 LEFT, TOP, RIGHT, BOTTOM |
| fill | 填充方式，是否在某个方向充满窗口。 值为垂直“Y”，水平“X”，水平+垂直“BOTH”或“NONE” |
| expand | 控件是否会随窗口缩放，“YES”是可扩展，“NO”是不可扩展 |

在 Tkinter 中, `pack_forget()` 方法和 `forget()` 方法用于删除 `pack()` 方法的控件显示, 但仅仅是控件不显示了, 旧的显示位置也会消除, 而控件对象没有删除。如果需要再显示控件, 就可以在新位置用 `pack()` 方法。

`pack_info()` 方法用于显示 `pack()` 方法的设置信息, 并返回有关 `pack()` 方法中参数的一个字典数据。

`pack_configure()` 方法用于配置 `pack()` 方法的控件。

下面给出 `pack()` 方法的演示, 见示例 8-9。

```
import tkinter as tk          # 装载 tkinter 模块, 用于 Python 3
from tkinter import ttk      # 装载 tkinter.ttk 模块, 用于 Python 3

root = tk.Tk()
root.title('pack() 演示')
root.geometry('100x100')

tk.Label(root, text='A', bg='red').pack(side=tk.LEFT, expand=1, fill=tk.Y)
tk.Label(root, text='B', bg='yellow').pack(side=tk.TOP, expand=1, fill=tk.BOTH)
tk.Label(root, text='C', bg='yellow').pack(side=tk.RIGHT, expand=1, fill=
tk.NONE, anchor=tk.NE)

tk.Label(root, text='D', bg='red').pack(side=tk.LEFT, expand=0, fill=tk.Y)
tk.Label(root, text='E', bg='red').pack(side=tk.TOP, expand=0, fill=tk.BOTH)
tk.Label(root, text='F', bg='yellow').pack(side=tk.BOTTOM, expand=1)
tk.Label(root, text='G', bg='green').pack(anchor=tk.SE)

root.mainloop()
```

程序运行结果见图 8-12。

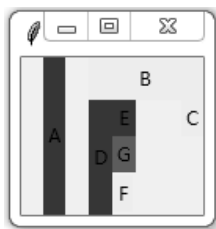


图 8-12 示例程序运行结果

2. Grid 布局

控件的 Grid 布局是网格布局, 是使用行列位置的坐标来放置控件的几何布局管理器, 其使用 `grid()` 方法。`grid()` 方法布局管理器会将控件放置到一个虚拟的二维表

格里，主控件被分割成一系列的行和列，表格中的每个单元都可以放置一个控件。

`grid()` 是 Tkinter 最为灵活的几何管理布局器，其不能和 `pack()` 方法混合使用。

当你设计对话框时，Grid 布局管理器是一个非常方便的工具。在大多数情况下，只要你将所需的控件放置到容器类控件中，然后使用 `grid()` 方法就能将它们布局到任何你想要布局的地方。

如果使用 Pack 进行布局，可能还需要一些额外的 Frame 控件，并需要自行调整。而如果你使用 Grid 进行布局，只需要对每个控件使用 `grid()` 方法，那么所有的控件就都会以合适的方式显示。

使用 `grid()` 方法显示控件 `w`，其语法格式如下。

```
w.grid(option=value, ...)
```

其中，option 的属性关键字及说明见表 8-3。

表 8-3 Grid 控件的属性关键字

| 关键字 | 说明 |
|------------|-------------------------------|
| column | 控件所在的起始列位置 |
| columnspan | 控件的列宽，即跨越列数 |
| in_ | in_ = w2 使用，将 w 注册为 w2 控件的子控件 |
| ipadx | 设置控件中水平方向空白区域的大小 |
| ipady | 设置控件中垂直方向空白区域的大小 |
| padx | x 轴间距 |
| pady | y 轴间距 |
| row | 控件所在的列起始行位置 |
| rowspan | 控件的行宽，即跨越行数 |
| sticky | 对齐方式: “N” “S” “E” “W” |

由于我们的程序大多数都是矩形的，因此特别适合于网格布局，即 Grid 布局。

在使用 `grid()` 方法时，至少要给出两个参数，即用 `row` 表示行、用 `column` 表示列，它们的编号都从 0 开始。

`grid()` 方法的另一个参数 `sticky`，可以用 “N” “E” “S” “W” 表示上右下左，它决定了控件开始的方向。

`grid()` 方法可以直接用后面的行和列的数字来指定它位于的位置，不必使用其他参数。

`grid_forget()`方法用于删除 `grid()`方法的控件显示，但仅仅是控件不显示了，旧位置也会消除，而控件对象没有删除。如果想显示控件，就可以在新位置用 `grid()`方法。

`grid_info()`方法用于显示 `grid()`方法的设置信息，返回关于 `grid()`方法中参数关键字的一个字典数据。

`grid_configure()`方法用于配置 `grid()`方法的控件。

见示例 8-10。

```
import tkinter as tk          #装载 tkinter 模块，用于 Python 3
from tkinter import ttk      #装载 tkinter.ttk 模块，用于 Python 3
root = tk.Tk()
root.title('grid() 演示')
root.geometry('140x140')
tk.Label(root, text='A', bg='red', width=3, height=2).grid(row=0, column=0)
tk.Label(root, text='B', bg='yellow').grid(row=0, column=1)
tk.Label(root, text='C', bg='yellow', height=3).grid(row=0, column=3,
rowspan=2)          #跨 2 列
tk.Label(root, text='D', bg='red', width=10, height=1).grid(row=4, column=0,
rowspan=4)
tk.Label(root, text='E', bg='green', width=5).grid(row=2, column=0,
columnspan=2)      #跨 2 行
tk.Label(root, text='F', bg='red', width=5).grid(row=1, column=3,
rowspan=2)
root.mainloop()
```

程序运行结果见图 8-13。

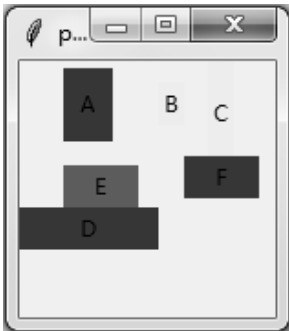


图 8-13 示例程序运行结果

3. Place 布局

Place 布局管理器可以显示指定控件的绝对位置或相对于其他控件的位置。如果想要使用 Place 布局，只需调用相应控件的 `place()` 方法就可以了，所有 Tkinter 的标准控件都可以调用 `place()` 方法。

使用 `place()` 方法显示控件 `w`，其语法格式如下。

```
w.place(option=value, ...)
```

其中，`option` 的属性关键字及说明见表 8-4。

表 8-4 Place 控件的属性关键字

| 关键字 | 说明 |
|-----------|---|
| anchor | 控件的对齐方式：顶对齐“N”，底对齐“S”，左对齐“W”，右对齐“E”。 值为 N, NE, E, SE, S, SW, W, NW, CENTER |
| x | 控件左上角的 x 坐标 |
| y | 控件左上角的 y 坐标 |
| relx | 控件左上角相对于窗口的 x 坐标，应为 0~1 的小数 |
| rely | 控件左上角相对于窗口的 y 坐标，应为 0~1 的小数 |
| width | 控件的宽度 |
| height | 控件的高度 |
| relwidth | 控件相对于窗口的宽度，0~1 的小数，图形宽度相对于窗口的变化 |
| relheight | 控件相对于窗口的高度，0~1 的小数，图形高度相对于窗口的变化 |

Place 布局分为绝对布局和相对布局，绝对布局使用 `x` 和 `y` 参数关键字，相对布局使用 `relx`，`rely`，`relheight` 和 `relwidth` 参数关键字。

如果窗口大小是固定的，就可以使用 `place()` 方法的绝对布局方式，否则窗口大小改变后就不好看了。

由于 `place()` 方法可以与 `pack()` 方法和 `grid()` 方法混用，因此使用 `place()` 方法布局的控件可以叠加。

下面给出一个 `place()` 方法的示例，见示例 8-11。

```
import tkinter as tk #装载tkinter模块，用于Python 3
root = tk.Tk()
root.title('place() 演示')
root.geometry('100x100')
tk.Label(root, text='A', bg='red').place(x=10, y=10)
```

```
tk.Label(root,text='B',bg='yellow').place(x=10, y=80)
tk.Label(root,text='C',bg='yellow').place(x=80, y=20)
tk.Label(root,text='D',bg='red').place(x=30, y=50)
tk.Label(root,text='E',bg='red').place(x=40, y=50)
tk.Label(root,text='F',bg='yellow').place(x=80, y=70)
tk.Label(root,text='G',bg='green').place(x=30, y=20)
root.mainloop()
```

程序运行结果见图 8-14。

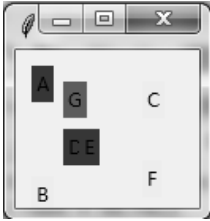


图 8-14 示例程序运行结果

8.8 Tkinter常用控件

Tkinter 常用控件及功能见表 8-5。

表 8-5 Tkinter 的控件

| 控件 | 功能 |
|-------------|---------------------------------------|
| Button | 按钮，在程序中显示按钮 |
| Canvas | 绘图形组件，可以在其中绘制图形 |
| Checkbutton | 复选框，用于在程序中提供多项选择框 |
| Entry | 文本框（单行），用于显示简单的文本内容 |
| Text | 文本框（多行），用于显示多行文本 |
| Frame | 框架，将几个组件组成一组。在屏幕上显示一个矩形区域，多用来作为容器 |
| Label | 标签，可以显示文字或图片 |
| Listbox | 列表框，在 Listbox 窗口的小部件，用来显示一组字符串列表给用户选择 |
| Menu | 菜单，显示菜单栏，包括下拉菜单和弹出菜单 |
| Menubutton | 菜单按钮控件，用于显示菜单项，可以使用 Menu 代替 |
| Message | 与 Label 组件类似，但是可以根据自身大小将文本换行 |

续表

| 控件 | 功能 |
|-------------|---|
| Radiobutton | 单选框，显示一个单选的按钮状态 |
| Scale | 滑块，允许通过滑块来设置数字值为输出限定范围的数字区间 |
| Scrollbar | 滚动条，配合使用 canvas, entry, listbox, text 等窗口控件的标准滚动条 |
| Toplevel | 用来创建子窗口组件 |

除了表格中列出的控件外，Tkinter 的控件还包含一些对话框模块，如 messagebox, filedialog, simpledialog, colorchooser 等。

messagebox: 为简单的任务提供七种常见的弹出式对话框。

filedialog: 允许用户浏览文件。

colorchooser: 允许用户选择颜色。

simpledialog: 允许用户输入整型数、浮点数、字符串的对话框。

下面简单介绍一下常见的 Tkinter 控件。

1. Label 控件

Label 控件是 Tkinter 最常用的控件之一，其语法格式如下。

```
label=tk.Label ( master, option, ... )
```

其中，master 为其父控件，就是用来放置这个控件的。

option 是属性关键字，其可以在控件创建时设置属性，也可以在创建控件后再为其指定属性，因此在创建方法中的 option 选项参数可以为空。

Label 控件的关键字及说明见表 8-6。

表 8-6 Label 控件的关键字

| 关键字 | 说明 |
|------------------|--|
| activebackground | 在设置活动状态时的背景色，默认值系统指定 |
| activeforeground | 在设置活动状态时的前景色，默认值系统指定 |
| anchor | 控制文本或者图像在 Label 中的显示位置，值为 N, NE, E, SE, S, SW, W, NW, CENTER。默认值为 CENTER |
| background 或 bg | 设置背景色 |
| foreground 或 fg | 设置前景色 |
| bitmap | 指定显示到控件上的位图。如果指定了 image，则该选项忽略 |
| borderwidth 或 bd | 指定边框宽度，默认值由系统指定，通常为 1 像素或者 2 像素 |

续表

| 关键字 | 说明 |
|---------------------|---|
| compound | <p>文本和图像混合模式。默认值是 NONE。</p> <p>在默认情况下，如果指定位图和图片，文本则不显示。</p> <p>如果选项设置为 CENTER，文本则显示在图像上。</p> <p>如果设置为 BOTTOM, LEFT, RIGHT, TOP，那么图像则显示在文本的旁边；如果设置为 BOTTON，则图像在文本的下方</p> |
| cursor | 指定当前鼠标在 Label 上飘过时的鼠标样式。默认系统指定 |
| disabledforeground | 指定 Label 在不可用时的前景色。默认系统指定 |
| font | 指定字体样式。默认由系统指定 |
| height | 设置 Label 的高度。如果 Label 是文本，单位则是文字高度，否则为像素。如果为 0 或者默认，则自动根据 Label 计算 |
| highlightbackground | 指定当 Label 没有获得焦点时的高亮边框颜色。默认系统指定 |
| highlightcolor | 当 Label 在获得焦点时的高亮边框颜色。默认系统指定 |
| highlightthickness | 指定高亮边框的宽度。默认值是 0 |
| image | 指定图像。该值应该是 PhotoImage, BitmapImage 或者能兼容的对象。该选项优先于参数关键字 text 和 bitmap |
| justify | 定义对齐方式，值为 LEFT, RIGHT, CENTER。文本位置取决于 anchor 选项，默认值为 CENTER |
| padx | x 轴间距，单位为像素 |
| pady | y 轴间距，单位为像素 |
| relief | 指定边框样式。默认值是 FLAT，可以设置为 SUNKEN, RAISED, GROOVE, RIDGE |
| state | 指定 Label 的状态。这个标签控制 Label 如何显示，默认值是 NORMAL，可设置为 ACTIVE 或 DISABLED |
| takefocus | 如果是 True，则该 Label 接收输入焦点。默认值为 False |
| text | 指定文本，文本可包含换行符。如果设置关键字参数 image 或 bitmap，则该选项被忽略 |
| textvariable | Label 显示 Tkinter 字符变量 StringVar。如果变量值被修改，Label 文本就自动更新 |
| underline | 与 text 选项一起使用，用户指定的字符画下画线。默认值是-1。如果设置为 1，则从第二个字符开始画下画线 |
| width | 设置 Label 的宽度。如果 Label 显示文本，则单位为文本长度，否则为像素。如果设置为 0 或者默认就自动计算 |
| wraplength | 决定了 Label 文本将被分成多少行。该选项指定每行的长度，单位是屏幕单元，默认值为 0 |

Tkinter 容器控件没有背景图片属性，通常都是利用 Label 图片属性来给容器控件增加背景图片。

下面是针对 Label 控件的一个简单操作，见示例 8-12，示例中窗口显示当前的日期时间。

```
import tkinter as tk                #装载tkinter模块，用于Python 3
import time
root = tk.Tk()
root.title(string = '最新时间')    #设置窗口标题
root.geometry('400x100')
img_png =tk.PhotoImage(file = 'b.png')
label_img = tk.Label(root, image = img_png)
label_img.place(x=0, y=0, relwidth=1, relheight=1,bordermode= tk.OUTSIDE)
running = False
timestr1 = tk.StringVar()
timestr2 = tk.StringVar()
l1 = tk.Label(root, textvariable = timestr1,bg='#1E488F',fg='yellow',
font = 'Helvetica -24 bold',width=20,height=2)
l2 = tk.Label(root, textvariable = timestr2,bg='#1E488F',fg='yellow',
font = 'Helvetica -20 bold')
l1.pack()
l2.pack()
def _settime():
    today1 = str('现在时间: '+time.strftime('%Y-%m-%d', time.localtime(
time.time()))))
    time1 = str('现在日期: '+time.strftime('%H:%M:%S', time.localtime(
time.time()))))
    timestr1.set(today1)
    timestr2.set(time1)
_settime()
root.mainloop()
```

程序运行结果见图 8-15。



图 8-15 示例程序运行结果

2. Frame 控件

Frame 控件是 Tkinter 的容器控件，负责安排其他控件的位置。Frame 控件采用在屏幕上的矩形区域组织布局，并提供其他控件和容器控件本身的填充。一个 Frame 控件也可以用来作为一个基础类，以实现复杂的构件。

小白量化投资分析软件框架使用了大量在 Frame 控件上建立的新控件，以实现不同的功能。

窗口画面布局过程：

首先，建立布局，即主窗口布局。用不同颜色和大小 Frame 类进行填充，调整位置直到自己满意。

其次，设计不同功能的子窗口，就是 Frame 类放在 Frame 中，如 Label，Button 等，并调整好控件的大小和位置。

最后，根据需要可以程序控制不同功能的 Frame 类显示或隐藏到对应的主窗口布局中。

Frame 语法格式：

```
frame = tk.Frame ( master, option, ... )
```

其中，master 为父窗口，option 是控件的属性选项关键字列表。

Frame 属性选项关键字及说明见表 8-7。

表 8-7 Frame 属性选项关键字

| 关键字 | 说明 |
|---------------------|--|
| bg 或 background | 设置 Frame 控件的背景颜色。默认值由系统指定 |
| bd 或 borderwidth | 指定 Frame 的边框宽度。默认值是 0 |
| cursor | 指定当鼠标在 Frame 上飘过时的鼠标样式。默认值由系统指定 |
| height | 设置 Frame 的高度。默认值是 0 |
| highlightbackground | 指定当 Frame 没有获得焦点时高亮边框的颜色。默认值由系统指定，通常是标准背景颜色 |
| highlightcolor | 指定当 Frame 获得焦点时高亮边框的颜色。默认值由系统指定 |
| highlightthickness | 指定高亮边框的宽度。默认值是 0（不带高亮边框） |
| padx | 水平方向上的边距 |
| pady | 垂直方向上的边距 |
| relief | 指定边框样式，默认值是 FLAT。另外，还可以设置为 SUNKEN, RAISED, GROOVE, RIDGE。 注意：如果你要设置边框样式，就记得设置为 borderwidth 或 bd，因为只有选项不为 0 才能看到边框 |
| takefocus | 指定该控件是否接受输入焦点（用户可以通过 tab 键将焦点转移上来）。默认值是 False |
| width | 设置 Frame 的宽度。默认值是 0 |

从 Frame 控件的属性参数表格中可以看到，其很多属性同 Label 控件的属性。Frame 控件演示见示例 8-13。

```
import tkinter as tk          #装载 tkinter 模块，用于 Python 3
root=tk.Tk()                  #创建 Tkinter 主窗口
root.title("Frame 演示")
labels=[]                     #标签列表
reliefs=[tk.FLAT,tk.SUNKEN,tk.RAISED,tk.GROOVE,tk.RIDGE]
i=0
for color in ['blue','yellow','red','purple','pink']:
    f = tk.Frame(root, borderwidth=1,bg=color,relief=reliefs[i],
                  bd=20,width=100,height=100,container=True)
    f.pack(side=tk.LEFT)
    i+=1
root.mainloop()               #进入消息循环
```

程序运行结果见图 8-16。

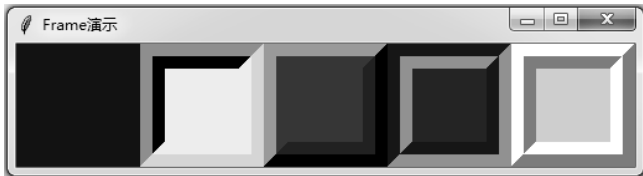


图 8-16 示例程序运行结果

3. Button 控件

Button 控件是一个标准的 Tkinter 控件，用于实现各种包含文本和图像的按钮。可以调用 Python 的函数用于每个按钮，在 Tkinter 的按钮被按下时，就会自动调用该函数。该按钮可以只显示在一个单一字体的文本中，但文本可能跨越一个以上的行。此外，一个字符可以有下画线，如标记的键盘快捷键。在默认情况下，使用 Tab 键可以移动到另一个按钮控件。Button 按钮方法支持 flash()方法和 invoke()方法。

Button 按钮通常用于应用程序窗口接受鼠标点击并执行相应的程序。

其语法格式如下：

```
button=tk.Button ( master, option=value, ... )
```

其中，master 为父窗口，option 是控件的属性关键字选项列表。

Button 控件的属性关键字及说明见表 8-8。

表 8-8 Button 控件的属性关键字

| 关键字 | 说明 |
|------------------|--|
| anchor | 指定按钮上文本的位置, 值为 N, NE, E, SE, S, SW, W, NW, CENTER, (EWSN 表示东西南北)。默认值为 CENTER |
| background 或 bg | 指定按钮的背景色 |
| Bitmap | 指定按钮上显示的位图 |
| borderwidth 或 bd | 指定按钮边框的宽度 |
| command | 指定按钮消息的回调函数 |
| cursor | 指定鼠标移动到按钮上的指针样式 |
| font | 指定按钮上文本的字体 |
| focus_set | 设置当前控件得到的焦点 |
| relief | 指定外观, 参数值有 FLAT, SUNKEN, RAISED, GROOVE, RIDGE |
| background 或 fg | 指定按钮的前景色 |
| height | 指定按钮的高度 |
| image | 指定按钮上显示的图片 |
| state | 指定按钮的状态, 值为 NORMAL, ACTIVE, DISABLED。默认值为 NORMAL |
| text | 指定按钮上显示的文本 |
| width | 指定按钮的宽度 |
| padx | 设置文本与按钮边框 x 的距离 |
| pady | 设置文本与按钮边框 y 的距离 |
| activeforeground | 按钮按下前景色 |
| textvariable | 可变文本, 与 StringVar 等配合使用 |
| underline | 在文本标签中确定哪个字符加下画线。默认值为-1, 意思是没有字符加下画线 |

下面主要介绍一下 Button 控件特有属性的用法:

command 属性指定按钮消息的回调函数, 即鼠标左键按下后, Button 按钮执行 command 属性指定的回调函数或方法。

state 属性指定按钮的状态。如果 state 的值为 DISABLED, 那么 Button 控件就是处于不可用状态, 即按钮无法点击, 也不会执行 command 属性指定的回调函数或方法。

提示: 针对 Button 控件 command 参数的回调函数, 如果函数不带参数, 就直接使用函数名; 如果函数带参数, 就需要使用 “lambda: function(参数表)” 的格式。

与 Label 控件不同, Button 控件除了有属性以外, 还有一些方法。

Button 控件的方法见表 8-9。

表 8-9 Button 控件的方法

| 方法 | 说明 |
|----------|---------------------------------|
| flash() | 在激活状态颜色和正常颜色之间闪烁几次，闪烁完成后，保持初始状态 |
| invoke() | 执行按钮按下的操作 |

flash()方法可以使某个按钮快速在普通和激活状态下闪烁几次。

invoke()方法可以在程序中模拟当某个按钮被按下时所执行的回调函数。

示例 8-14 给出一些 Button 的演示，其中在鼠标移动到不同的 Button 时，鼠标的形状也会改变。

```
import tkinter as tk #装载tkinter模块,用于Python3

def bb():
    print('hello !')

root = tk.Tk()
root.title('Button 测试')

tk.Button(root,text='设置 bitmap 放置到按钮左边位置', compound='left',
bitmap='error',cursor='heart').pack()
tk.Button(root,text='设置 command 事件调用命令', width=28,fg='blue',bd=2,
command=bb,cursor='icon',underline=4).pack()
tk.Button(root,text='外观装饰边界附近的标签', width=19,relief=tk.GROOVE,
bg='yellow',cursor='hand2').pack()
tk.Button(root,text='设置按钮状态', width=21,state=tk.DISABLED,
cursor='mouse').pack()
tk.Button(root,text='设置高度宽度以及文字显示位置',anchor=tk.SW,width=30,
height=2,cursor='man').pack()

root.mainloop()
```

示例演示结果见图 8-17。

4. Entry 控件

Entry 控件是 Tkinter 的文本框，是 Tkinter 用来接收字符串等输入的控件。该控件允许用户查看和修改一行文本，但是当用户输入的文字长度大于 Entry 控件的宽度时，文字就会向后滚动。这种情况下所输入的字符串就无法全部显示，可以点击

箭头符号将不可见的文字部分移入到可见区域。如果你想要输入多行文本，就需要使用 Text 控件。另外，Entry 控件只能使用预设字体。



图 8-17 示例程序运行结果

使用 Entry 控件的语法如下：

```
entry = tk.Entry (master, option, ... )
```

其中，master 为父窗口，option 是控件的属性关键字选项列表。

Entry 控件的属性关键字及说明见表 8-10。

表 8-10 Entry 控件的属性关键字

| 关键字 | 说明 |
|---------------------|--|
| background 或 bg | 指定控件的背景色 |
| borderwidth 或 bd | 指定控件边框的宽度 |
| cursor | 指定鼠标移动到控件的指针样式 |
| disabledbackground | 当控件在禁用状态时要显示的背景颜色 |
| disabledforeground | 当控件在禁用状态时要显示的前景颜色 |
| exportselection | 在默认情况下，如果在输入框中选中文本，默认就会复制到粘贴板；如果要忽略这个功能，就设置为 exportselection=0 |
| foreground 或 fg | 指定控件的前景色 |
| font | 指定控件上文本的字体 |
| highlightbackground | 文本框高亮边框颜色，当文本框未获取焦点时显示 |
| highlightcolor | 文本框高亮边框颜色，当文本框获取焦点时显示 |
| highlightthickness | 文本框高亮边框宽度 |
| insertbackground | 文本框光标的颜色 |
| insertborderwidth | 文本框光标的宽度 |

续表

| 关键字 | 说明 |
|--------------------|---|
| insertofftime | 在文本框光标闪烁时, 消失持续的时间, 单位为毫秒 |
| insertontime | 在文本框光标闪烁时, 显示持续的时间, 单位为毫秒 |
| insertwidth | 文本框光标宽度 |
| justify | 在显示多行文本时, 可以通过 LEFT, RIGHT, CENTER 设置不同行之间的对齐方式 |
| readonlybackground | 当控件在只读状态时的背景颜色 |
| relief | 指定控件外观。参数值有 FLAT, SUNKEN, RAISED, GROOVE, RIDGE 等 |
| selectbackground | 选中文字的背景颜色 |
| selectborderwidth | 选中文字的背景边框宽度 |
| selectforeground | 选中文字的颜色 |
| show | 指定文本框内容显示为字符, 值随意, 如密码可以将值设为* |
| state | 文本框状态, 分为只读和可写。值为 NORMAL, ACTIVE, DISABLED 等, 默认值为 NORMAL |
| takefocus | 是否能用 TAB 键来获取焦点, 默认是可以获得 |
| textvariable | 文本框的值, 是一个 StringVar()对象, 与 StringVar 等配合使用 |
| width | 指定控件的宽度, 设置控件中允许显示的字符数。默认值为 20 |
| xscrollcommand | 使用此选项可将 Scrollbar 控件连接到这个控件 |

Entry 控件还提供一些方法可用, 见表 8-11。

表 8-11 Entry 控件方法

| 方法 | 说明 |
|------------------------------------|-----------------------------|
| insert | 向文本框中插入值 |
| delete | 删除 |
| icursor | 将光标移动到指定索引位置, 只有当文本框获取焦点后成立 |
| get | 获取文件框的值 |
| index | 返回指定的索引值 |
| selection_adjust 或 select_adjust | 选中指定索引和光标所在位置之前的值 |
| selection_clear 或 select_clear | 清空文本框 |
| selection_from 或 select_from | 设置光标的位置, 通过索引值 index 来设置 |
| selection_present 或 select_present | 如果选中, 就返回 true, 否则返回 false |
| selection_range 或 select_range | 选中范围 |
| selection_to, select_to | 选中指定索引与光标之间的值 |
| scan_mark | 查找 mark |

续表

| 方法 | 说明 |
|-----------------------------|---|
| scan_dragto | 查找 dragto |
| xview | 该方法在文本框链接到水平滚动条上时很有用 |
| xview_moveto 或 xview_scroll | 用于水平滚动文本框。what 参数可以是 UNITS，按字符宽度滚动；也可以是 PAGES，按文本框控件块滚动。number 参数的正数由左到右滚动，负数由右到左滚动 |

下面介绍两种常用的方法：

insert(index, text)是向文本框中插入值，index 是插入位置，text 是插入值。

get()是获取文件框的值。

见示例 8-15。

| | |
|---------------------------------------|--------------------|
| import tkinter as tk | #导入 Tkinter3 模块 |
| root =tk.Tk() | #创建 Tkinter 窗口对象 |
| root.title(string = '主窗口') | #设置窗口标题 |
| root.geometry('200x100') | #改变窗口大小 |
| user=tk.StringVar(root,'admin') | #建立 StringVar 变量 |
| User=tk.Entry(root,textvariable=user) | #创建 Entry 组件 |
| User.pack() | |
| User.insert(tk.END, 'istrator') | #在文本框末尾插入 istrator |
| u=User.get() | #获取 Entry 控件值 |
| print('用户名: ',u) | |
| root.mainloop() | #进入 Tkinter 消息循环 |

程序运行结果见图 8-18。

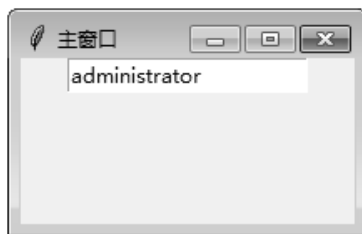


图 8-18 示例程序运行结果

5. Text 控件

Text 控件是 Tkinter 的多行文本编辑框，其十分强大且具有很强的灵活性，适用

于多种任务。虽说 Text 的主要作用是显示多行文本，但也常常被当作简单的文本处理器、文本编辑器来使用。

使用 Text 文本控件的语法如下：

```
text= tk. Text (master, option, ... )
```

其中，master 为父控件，option 是控件的属性关键字选项列表。

Text 的属性关键字及说明见表 8-12。

表 8-12 Text 的属性关键字

| 关键字 | 说明 |
|---------------------|--|
| autoseparators | 如果设置了 undo 选项，自动调整程序选项就会控制每次插入或插入后的分隔符自动添加到撤销堆栈中。删除为“autoseparators=True”或不删除为“autoseparators=False” |
| background 或 bg | 文本控件的默认背景颜色 |
| borderwidth 或 bd | 文本控件周围边框的宽度。默认是 2 像素 |
| cursor | 当鼠标位于文本控件上时会出现的指针样式 |
| exportselection | 在默认情况下，如果在文本框中选中文本，就会默认复制到粘贴板；如果要忽略这个功能，就需要设置“exportselection=0” |
| foreground 或 fg | 控件中用于文本（和位图）的颜色。你可以改变标记区域的颜色，此选项只是默认选项 |
| font | 插入到控件中的文本的默认字体。注意：你可以通过使用标记来更改某些文字的字体 |
| height | 指定行数。根据当前字体大小，以文字行为单位，注意不是以像素为单位 |
| highlightbackground | 文本框高亮边框颜色，当文本框未获取焦点时显示 |
| highlightcolor | 文本框高亮边框颜色，当文本框获取焦点时显示 |
| highlightthickness | 文本框高亮边框宽度 |
| insertbackground | 文本框光标的颜色 |
| insertborderwidth | 文本框光标的宽度 |
| insertofftime | 在文本框光标闪烁时，消失持续的时间，单位为毫秒 |
| insertontime | 在文本框光标闪烁时，显示持续的时间，单位为毫秒 |
| insertwidth | 文本框光标宽度 |
| maxundo | 当此选项设置撤销时保留操作的最大数目，可将此选项设置为-1 以指定一个无限制撤销堆栈中的数目 |
| padx | 添加到文本区域的左侧和右侧的内部填充的大小。默认是 1 像素 |
| pady | 在文本区域的上方和下方添加的内部填充的大小。预设值是 1 像素 |
| relief | 指定控件外观。参数值有 FLAT, SUNKEN, RAISED, GROOVE, RIDGE 等 |

续表

| 关键字 | 说明 |
|-------------------|---|
| selectbackground | 显示选定文本的背景颜色 |
| selectborderwidth | 选定文本周围要使用的边框宽度 |
| selectforeground | 用于显示选定文本的前景颜色 |
| spacing1 | 此选项指定将多少额外的垂直空间置于文字。如果线条环绕，则仅在它所占用的第一行之前添加此空间在展览上。默认为 0 |
| spacing2 | 此选项指定要在显示之间添加多少额外的垂直空间文字。当一个逻辑行环绕时，将出现一行文本。默认为 0 |
| spacing3 | 此选项指定在每行下方添加多少额外的垂直空间文字。如果线条环绕，则仅在它所占的最后一行之后添加此空间在展览上。默认为 0 |
| state | 文本框状态，分为只读和可写，值为 NORMAL，ACTIVE，DISABLED。默认值为 NORMAL |
| tabs | 此选项控制制表符如何定位文本 |
| takefocus | 是否能用 TAB 键来获取焦点，默认是可以获得 |
| undo | 将此选项设为 True 以启用撤销机制，False 为禁用它 |
| width | 指定控件的宽度，即设置控件中允许的显示的字符数。默认值为 20 |
| wrap | 此选项控制显示过宽的行。 默认 wrap=CHAR，任何行过长的单词都会被破坏。 如果设置 wrap=WORD，它就会在最后一个合适的单词之后断线。 如果设置 wrap=NONE，不换行，就可以使用水平滚动条移动 |
| xscrollcommand | 使用此选项可将 Scrollbar 控件连接到此控件的 x 轴 |
| yscrollcommand | 使用此选项可将 Scrollbar 控件连接到此控件的 y 轴 |

Text 控件也有很多操作方法，下面介绍几种简单的。

1) Text 控件插入

为了给 Text 控件插入内容，可以使用 insert()方法及“INSERT”或“END”索引号，也可以按行列索引插入，格式为“行号.列号”。注意，行号从 1 开始，列号从 0 开始。见示例 8-16。

```
import tkinter as tk
root = tk.Tk()
text1 = tk.Text(root,width=30,height=4)
#INSERT 索引表示在光标处插入
text1.insert(tk.INSERT,'插入到中间。')
#END 索引号表示在最后插入
```



```
text1.insert(tk.END, ' 插入在最后。')
text1.pack()
root.mainloop()
```

2) 清除 Text 控件的内容

清除 Text 控件的所有内容的格式如下:

```
Text1.delete(1.0,tk.END)
```

3) 获取 Text 控件的内容

获取 Text 控件的所有内容的格式如下:

```
T1=Text1.get(1.0,tk.END)
```

4) 在 Text 控件中插入图片

在 Text 控件中插入图片的格式如下:

```
#装入图片文件
photo = PhotoImage(file='cat.gif')
#添加图片用 image_create
text1.image_create(END,image=photo)
#添加插件用 window_create
text1.window_create(INSERT>window=b1)
```

5) 在 Text 控件中插入 Button 按钮

在 Text 控件中插入 Button 按钮的格式如下:

```
def show():
    print('哎呀,我被点了一下')
#Text 中还可以插入按钮
b1 = Button(text1,text='点我吧',command=show)
#在 Text 中创建控件的命令
text1.window_create(INSERT>window=b1)
```

6) Text 控件的 Tags 用法

标签“Tags”通常用于改变 Text 控件中内容的样式和功能,即可修改文本的字体、尺寸和颜色。另外,Tags 还可将文本、嵌入的控件、图片与鼠标和键盘等事件相关联。

Tags 可以自定义任意数量的、其名字由普通字符串组成的,除了空白字符以外

的任何字符。另外，任何文本内容都支持多个 Tags 描述，且任何 Tags 也可以用于描述多个不同的文本内容。

为指定文本添加 Tags 可以使用 `tag_add()` 方法，设置 Tags 的样式可以使用 `tag_config()` 方法。下面是 `tag_config()` 方法可以使用的选项，见表 8-13。

表 8-13 `tag_config()` 方法可以使用的选项

| 选项 | 含义 |
|-------------|---|
| background | 指定该 Tag 所描述的内容的背景颜色。 注意：bg 并不是该选项的缩写，而是 bgstipple 选项的缩写 |
| bgstipple | ①指定一个位图作为背景，并使用 background 选项指定的颜色填充。 ②只有设定了 background 选项，该选项才会生效。 ③默认的标准位图字符串有 error, gray75, gray50, gray25, gray12, hourglass, info, questhead, question, warning 等 |
| borderwidth | ①指定文本框的宽度。 ②默认值是 0。 ③只有设定了 relief 选项，该选项才会生效。 注意：该选项不能使用“bd”缩写 |
| fgstipple | ①指定一个位图作为前景色。 ②默认的标准位图字符串有 error, gray75, gray50, gray25, gray12, hourglass, info, questhead, question, warning 等 |
| font | 指定该 Tag 所描述的内容使用的字体 |
| foreground | 指定该 Tag 所描述的内容使用的前景色。 注意：fg 不是该选项的缩写，而是 fgstipple 的缩写 |
| justify | ①控制文本的对齐方式。 ②默认是 LEFT（左对齐），还可以选择 RIGHT（右对齐）和 CENTER（居中）。 注意：需要将 Tag 指向该行的第一个字符，该选项才能生效 |
| lmargin1 | ①设置 Tag 指向的文本块第一行缩进。 ②默认值是 0。 注意：需要将 Tag 指向该行的第一个字符或整个文本块，该选项才生效 |
| lmargin2 | ①设置 Tag 指向的文本块，除第一行外其他行的缩进。 ②默认值是 0。 注意：需要将 Tag 指向整个文本块，该选项才能生效 |
| offset | ①设置 Tag 指向的文本相对于基线的偏移距离。 ②可以控制文本相对于基线是升高（正数值）或者降低（负数值）。 ③默认值是 0 |

续表

| 选项 | 含义 |
|------------|--|
| overstrike | ①在 Tag 指定的文本范围内画一条删除线。 ②默认值是 False |
| relief | 指定 Tag 对应范围的文本的边框样式。 ②可以使用的值有 SUNKEN, RAISED, GROOVE, RIDGE, FLAT 等。 ③默认值是 FLAT (没有边框) |
| margin | ①设置 Tag 指向的文本块右侧的缩进。 ②默认值是 0 |
| spacing1 | 设置 Tag 所描述的文本块中每一行与上方的文本间隔, 自动换行不算。 ②默认值是 0 |
| spacing2 | 设置 Tag 所描述的文本块中自动换行的各行间的空白间隔, 换行符 (\n) 不算。 ②默认值是 0 |
| spacing3 | 设置 Tag 所描述的文本块中每一行与下方的文本间隔, 自动换行不算。 ②默认值是 0 |
| tabs | ①定制 Tag 所描述的文本块中 Tab 按键的功能。 ②默认 Tab 被定义为 8 个字符的宽度。 ③可定制多个制表位, 如 tabs=('3c', '5c', '12c')表示前三个 Tab 的宽度分别为 3cm, 5cm, 12cm, 接着的 Tab 按照最后两个的差值计算, 即 19cm, 26cm, 33cm。 ④应该注意到, 它上边 c 的含义是“厘米”而不是“字符”, 还可以选择的单位有 i (英寸), m (毫米), p (DPI, 大约是 1i 等于 72p)。 ⑤如果是一个整型值, 则单位是像素 |
| underline | ①如果该选项设置为 True, 则 Tag 所描述的范围内的文本将被画上下画线。 ②默认值是 False |

如果同一个范围内的文本加上多个 Tags (会形成一个栈的形式), 并且设置相同的选项, 那么新建的 Tags 样式就会覆盖旧的 Tags。

另外, Tags 还支持事件绑定, 使用的是 tag_bind()方法。

tag_add()方法的参数通常有四个, 具体如下:

第一个参数为自定义标签的名字。

第二个参数为设置的起始位置。

第三个参数为结束位置。

第四个参数为另一个位置。

下边的例子中, 我们将文本 Baidu.com 与鼠标事件进行绑定。当鼠标进入该文本段时, 鼠标样式切换为 arrow 形态; 当鼠标离开文本段时, 则切换回 xterm 形态。

当触发鼠标“左键点击操作”时，使用默认浏览器打开百度首页。见示例 8-17。

```
import tkinter as tk #装载 tkinter 库，用于 Python 3
import webbrowser
root = tk.Tk()
root.geometry('300x200')
text1 = tk.Text(root, width=30, height=10)
text1.pack()
def showmess():
    text1.insert(tk.END, '\n 哎呀，我被点了一下') #插入到 Text1 最后面
#用 tag_config 函数来设置标签的属性
text1.tag_config('tag1', background='white', foreground='red') #tag1 设置
text1.tag_config('tag2', background='white', foreground='blue')
#tag2 设置同 tag1
text1.tag_config('link', foreground='blue', underline=True) #link 设置

#text 还可以插入按钮
button1 = tk.Button(text1, text='来点我吧', command=showmess)
#在 text 中创建控件的命令
text1.insert(tk.INSERT, 'Tkinter Text 控件学习\n')
#text 还可以插入按钮
button1 = tk.Button(text1, text='来点我吧', command=showmess)
text1.window_create(tk.INSERT, window=button1)
text1.insert(tk.INSERT, '\nBaidu.com 百度的网址')
text1.insert(tk.INSERT, '\nI Love Python! I Love Python!')
text1.insert(tk.INSERT, '\n野火烧不尽，春风吹又生!')
text1.tag_add('tag1', '4.7', '4.13', '5.1')
text1.tag_add('tag2', '1.1', '1.16')
def clickme(event):
    webbrowser.open('http://www.baidu.com')
def show_hand2_cursor(event):
    text1.config(cursor='hand2')
def show_xterm_cursor(event):
    text1.config(cursor='xterm')
text1.tag_add('link', '3.0', '3.9')
text1.tag_config('link', foreground='blue', underline=True)
text1.tag_bind('link', '<Enter>', show_hand2_cursor)
text1.tag_bind('link', '<Leave>', show_xterm_cursor)
```

```
text1.tag_bind('link', '<Button-1>', clickme)
root.mainloop()
```

程序运行结果，我多次点击按钮增加了很多行信息，见图 8-19。



图 8-19 示例程序运行结果

6. Scrollbar 控件

Tkinter 的 Scrollbar 滚动条控件用于滚动一些控件的可见范围。Scrollbar 控件通常与 Text 控件、Canvas 控件、Listbox 控件一起使用，水平滚动条还能与 Entry 控件配合，通过滚动条滑动窗口可以进入更大的虚拟空间。

说明：根据方向不同，Scrollbar 控件可分为垂直滚动条和水平滚动条。具有上下和左右两个方向滚动功能的控件，如 Canvas 控件和 Text 部件，有水平和垂直两种滚动条。

使用 Scrollbar 滚动条控件的语法如下：

```
scrollbar=tk.Scrollbar(master, option, ... )
```

其中，master 为父控件，option 是控件的属性关键字选项列表。

Scrollbar 属性关键字及说明列表见表 8-14。

表 8-14 Scrollbar 属性关键字

| 关键字 | 说明 |
|------------------|--|
| activebackground | 鼠标在滑块和箭头上方的颜色 |
| activerelief | 当鼠标在滑块上方时，滑块显示不同的样式。在默认情况下，滑块以 RAISED 样式显示 |
| bg 或 background | 当鼠标不在滑块上时，滑块和箭头的颜色 |
| bd 或 borderwidth | 整个滑槽周围三维边界的宽度，以及箭头和滑块上三维效果的宽度，默认为 bd 或 borderwidth，滑槽周围没有边界，箭头周围有 2 像素的边界和滑块 |

续表

| 关键字 | 说明 |
|---------------------|---|
| command | 当滚动条移动时要调用的过程 |
| cursor | 当鼠标在滚动条上方时出现的鼠标样式。默认系统指定 |
| elementborderwidth | 箭头和滑块周围边界的宽度，预设边界宽度为-1 |
| highlightbackground | 当滚动条没有焦点时，滚动条的颜色成高亮色 |
| highlightcolor | 当滚动条有焦点时，滚动条的颜色会突出 |
| highlightthickness | 焦点的厚度突出，默认为 1。当设置为 0 时可以抑制显示焦点的亮点 |
| jump | 此选项控制当用户拖动滑块时发生的情况。在正常情况下，jump=0，即滑块的每个小拖动都会导致回调函数。如果将此选项设置为 1，则直到用户释放鼠标按钮才调用回调函数 |
| orient | “orient=HORIZONTAL”为水平滚动条，“orient=VERTICAL”为垂直滚动条（默认方向） |
| relief | 控制控件的样式，默认是 SUNKEN，这个选项在 Windows 中不起作用。样式有 FLAT, SUNKEN, RAISED, GROOVE, RIDGE 等 |
| repeatdelay | 控制按钮在槽槽里的时间。在滑块开始朝那个方向重复移动之前，默认是 repeatdelay=300，单位为毫秒 |
| repeatinterval | 当控制按钮被压在槽里时滑块移动的频率。默认是 repeatinterval=100，单位是毫秒 |
| takefocus | 是否能用键盘来获取焦点。默认是 1，即可以通过滚动条控件来标记焦点，通过光标键来移动滚动条 |
| troughcolor | 槽的颜色 |
| width | 设置滚动条的宽度 |

在某个控件上安装垂直滚动条的做法：

- (1) 设置该控件的“yscrollbar”的“command”选项为 Scrollbar 控件的 set()方法。
- (2) 设置 Scrollbar 控件的“command”选项为该控件的 yview()方法。

当控件的可视范围发生改变时，控件通过调用 set()方法通知 Scrollbar 控件，而当用户操纵滚动条时，将自动调用控件的 yview()方法。

添加水平滚动条只需要将 yscrollcommand 改为 xscrollcommand，yview 改为 xview 即可。

见示例 8-18。

```
import tkinter as tk #装载tkinter模块，用于Python 3
root = tk.Tk()
S = tk.Scrollbar(root)
```

```

T = tk.Text(root, height=4, width=50)
S.pack(side=tk.RIGHT, fill=tk.Y)
T.pack(side=tk.LEFT, fill=tk.Y)
S.config(command=T.yview)
T.config(yscrollcommand=S.set)
quote = """ Tk/Tcl has long been an integral
part of Python. It provides a robust and
platform independent windowing toolkit,
that is available to Python programmers
using the tkinter package, and its extension,
the tkinter.tix and the tkinter.ttk modules."""
T.insert(tk.END, quote)
root.mainloop()

```

程序运行结果见图 8-20。

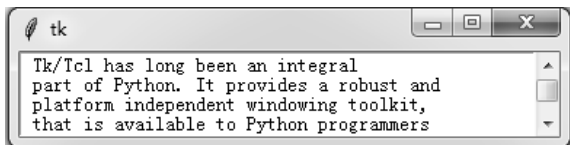


图 8-20 示例程序运行结果

7. Canvas 控件

Canvas 是 Tkinter 的画布控件，通常用于显示和编辑图形，即绘制线段、圆形、多边形，甚至是其他控件。

Canvas 控件的语法格式如下：

```
w = Canvas ( master, option=value, ... )
```

其中，master 为父控件，option 是控件的属性关键字选项列表。

Canvas 画布控件的属性关键字及说明见表 8-15。

表 8-15 Canvas 画布控件属性关键字

| 属性 | 说明 |
|----|--------------------|
| bd | 边框宽度，单位像素，默认为 2 像素 |
| bg | 背景色 |

续表

| 属性 | 说明 |
|------------------|---|
| confine | 如果为 True（默认），则画布不能滚动到可滑动的区域外 |
| cursor | 光标的形状设定，如 arrow，circle，cross，plus 等 |
| height | 画布在 y 坐标轴上的大小 |
| highlightcolor | 要高亮的颜色 |
| relief | 边框样式,可选值为 FLAT，SUNKEN，RAISED，GROOVE，RIDGE 等。默认为 FLAT |
| scrollregion | 元组 tuple (w, n, e, s)定义了画布可滚动的最大区域，其中 w 为左边，n 为头部，e 为右边，s 为底部 |
| width | 画布在 x 坐标轴上的大小 |
| xscrollincrement | 用于滚动请求水平滚动的数量值 |
| xscrollcommand | 水平滚动条，如果画布是可滚动的，则该属性是水平滚动条的.set()方法 |
| yscrollincrement | 类似于 xscrollincrement，但是为垂直方向 |
| yscrollcommand | 垂直滚动条，如果画布是可滚动的，则该属性是垂直滚动条的.set()方法 |

Canvas 控件支持画图对象及说明见表 8-16。

表 8-16 Canvas 控件

| 对象 | 说明 |
|-----------|--------------------------------|
| arc | 弧形、弦或扇形 |
| bitmap | 内建的位图文件或 XBM 格式的文件 |
| image | BitmapImage 或 PhotoImage 的实例对象 |
| line | 线 |
| oval | 圆或椭圆形 |
| polygon | 多边形 |
| rectangle | 矩形 |
| text | 文本 |
| window | window 控件 |

其中，弦、扇形、椭圆形、圆形、多边形和矩形这些“封闭式”图形都是由轮廓线和填充颜色组成的，但都可以设置为透明。

Canvas 控件对画图对象的操作见表 8-17。

表 8-17 Canvas 控件对画图对象的操作

| 操作 | 说明 |
|--------------|---------|
| coords() | 修改坐标和大小 |
| itemconfig() | 修改属性 |
| move() | 移动 |
| delete() | 删除 |

下面创建一个可在 Canvas 上手动绘图的应用，但 Canvas 并未提供画单个点的方法。我们可以通过绘制小的 oval 图形来解决这个问题，见示例 8-19。

```
import tkinter as tk                                #装载tkinter模块，用于Python 3
canvas_width = 300
canvas_height = 200
def checkered(canvas, line_distance):
    for x in range(line_distance, canvas_width, line_distance):
        canvas.create_line(x, 0, x, canvas_height, fill="#a0a0a0")
    for y in range(line_distance, canvas_height, line_distance):
        canvas.create_line(0, y, canvas_width, y, fill="#a0a0a0")

def paint( event ):
    python_green = "black"
    x1, y1 = ( event.x - 1 ), ( event.y - 1 )
    x2, y2 = ( event.x + 1 ), ( event.y + 1 )
    canvas.create_oval( x1, y1, x2, y2, fill = python_green )

master = tk.Tk()
master.title( "鼠标画图" )
canvas = tk.Canvas(master,
                    width=canvas_width,
                    height=canvas_height)
canvas.pack(expand = tk.YES, fill = tk.BOTH)
canvas.bind( "<B1-Motion>", paint )    #绑定鼠标左键
checkered(canvas,20)
message = tk.Label( master, text = "按下鼠标左键画图" )
message.pack( side = tk.BOTTOM )
master.mainloop()
```

程序运行结果见图 8-21。

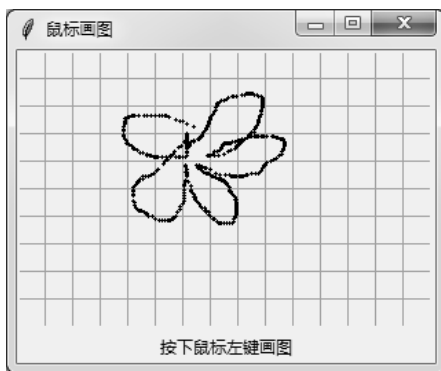


图 8-21 示例程序运行结果

8. 工具条和状态栏

由于 Tkinter 没有工具条的控件，因此可用 Frame 来代替工具条的布局。工具条的设计过程如下：

- (1) 建立一个 Frame。
- (2) 在 Frame 中放置几个 Button，靠左顺序排放。
- (3) 把设计好的 Frame 放置到窗口的最顶端，且在 x 轴方向填充。

Tkinter 也没有状态栏控件，可用 Frame 来建立状态栏，设计原理同工具条。这里给出了状态栏 StatusBar 的类，读者可以直接使用。

见示例 8-20。

```
import tkinter as tk #装载tkinter模块，用于Python 3
#状态栏
class StatusBar(tk.Frame):
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        tk.Label(self, bd=1, relief=tk.SUNKEN, anchor=tk.W, text=
'0000').place(x=0, y=0, relwidth=1, bordermode=tk.OUTSIDE)
        self.m=6 #有6个子栏
        self.l=[]
        self.l1 = tk.Label(self, bd=1, relief=tk.SUNKEN, anchor=
tk.CENTER, width=7, text='状态栏', justify=tk.CENTER)
        self.l1.pack(side=tk.LEFT, padx=1, pady=1)
```



```

        self.l1.append(self.l11)
        self.l12 = tk.Label(self, bd=1, relief=tk.SUNKEN, anchor=
tk.W,width=20,text='2')
        self.l12.pack(side=tk.LEFT,padx=1,pady=1)
        self.l1.append(self.l12)
        self.l13 = tk.Label(self, bd=1, anchor=tk.W,relief=
tk.SUNKEN,text='3')
        self.l13.pack(side=tk.LEFT,fill=tk.X)
        self.l1.append(self.l13)
        self.l14 = tk.Label(self, bd=1, relief=tk.SUNKEN, anchor=
tk.W,width=10,text='6')
        self.l14.pack(side=tk.RIGHT,padx=1,pady=1)
        self.l15 = tk.Label(self, bd=1, relief=tk.SUNKEN, anchor=
tk.W,width=10,text='5')
        self.l15.pack(side=tk.RIGHT,padx=1,pady=1)
        self.l16 = tk.Label(self, bd=1, relief=tk.SUNKEN, anchor=
tk.W,width=10,text='4')
        self.l16.pack(side=tk.RIGHT,padx=2,pady=1)
        self.l1.append(self.l16)
        self.l1.append(self.l15)
        self.l1.append(self.l14)
    def text(self,i,t):                                #输出文字信息
        self.l[i].config(text=t)
        self.l[i].update_idletasks()
    def config(self,i,**kargs):                        #配置长度和颜色
        for x,y in kargs.items():
            if x=='text':
                self.l[i].config(text=y)
            if x=='color':
                self.l[i].config(fg=y)
            if x=='width':
                self.l[i].config(width=y)
    def clear(self):                                    #清除所有信息
        for i in range(0,self.m):
            self.l[i].config(text='')
            self.l[i].update_idletasks()
    def set(self,i, format, *args):                    #输出格式信息

```

```

        self.l[i].config(text=format % args)
        self.l[i].update_idletasks()
root = tk.Tk()
root.geometry('800x400')
def callback():
    print ("clicked tool bar button ")
toolbar = tk.Frame(root,bg='#ACACAC')
    #由于tkinter 没有工具条的类，因此用 Frame 来代替工具条的布局
b = tk.Button(toolbar,text='Flie',width=6,command=callback)
b.pack(side=tk.LEFT,padx=2,pady=2)      #放在容器最左面
c = tk.Button(toolbar,text='Edit',width=6,command=callback)
c.pack(side=tk.LEFT,padx=2,pady=2)      #放在容器最左面
toolbar.pack(side=tk.TOP,fill=tk.X)      #放在最上面
status=StatusBar(root)
status.pack(side=tk.BOTTOM, fill=tk.X)
#status.clear()
status.text(1,'这里是短提示信息')        #在状态栏 2 输出信息
status.text(2,'这里是输出长信息aaaaaa')
status.config(1,color='red')              #改变状态栏 2 信息颜色
status.config(5,width=5)                  #改变状态栏 6 的宽度
status.set(3,'CPU: %0.2f'%12.5)           #格式输出信息
root.mainloop()

```

程序运行结果见图 8-22。



图 8-22 示例程序运行结果

8.9 Tkinter的事件和绑定

事件是发生在应用程序上的事情，多数情况是对用户按键和鼠标操作做出的反应，包括用户间接引起的事件（如控件状态改变）和程序上、系统上的自动事件（如定时器事件）。

Tkinter 应用程序将大部分时间花在通过 `mainloop()` 方法启动的事件循环中。Tkinter 提供了一种强大的事件处理机制，既可以自己处理事件也可以将 Python 的函数和方法绑定到事件上。

Tkinter 控件通常有很多内置的行为。例如，按钮通过调用回调命令对鼠标单击做出反应。Tkinter 的事件绑定功能允许添加、更改或删除新的事件处理能力。例如，前面出现的 `w.bind("<B1-Motion>", paint)` 代码给控件“w”绑定鼠标左键按下处理程序。

Tkinter 的应用程序需要知道发生的一些事件才能进行事件处理。事件处理程序是在事件发生时应用程序调用的函数。当应用程序设置事件处理程序时，我们将其称为绑定在一个部件上。

事件绑定有三种方式：

(1) 通过“`command = 回调函数`”，连接事件，按钮级别。

例如，`b1=tk.Button(root,text="button", command=hello2)`。

(2) 通过 `bind()` 方法绑定事件，控件级别。

例如，`w.bind("<B1-Motion>", paint)`。

(3) 通过 `protocol()` 方法来绑定系统级别事件。例如，可以安装其他协议的处理程序，如 `WM_TAKE_FOCUS` 和 `WM_SAVE_YOURSELF`。

例如，`root.protocol("WM_DELETE_WINDOW",callback)`。

我们通常使用 `bind()` 方法绑定事件，第一个参数是事件名称，第二个参数是事件回调函数。

下面我们看看事件名称的构成和编写事件的处理程序。

1. 事件名称

一般来说，事件名称是包含一个或多个事件模式的字符串，而每个事件模式都描述了可能发生的一件事。如果序列中有多个事件模式，则只有当所有模式都发生在同一序列中时才会调用处理程序。

事件名称的一般形式：

```
<[modifier-]...type[-detail]>
```

整个模式都包含在<...>内，其中参数的含义如下。

type：描述普通的事件类型，比如鼠标的单击或者键盘的单击。

modifier：修饰项，可选，通常用于描述组合键。

detail：事件名称，可选，用于描述具体的按键。

事件类型描述了事件的一般类型，如按键或鼠标单击。

在类型之前可以添加可选的修饰项以指定组合，如在其他键按下或鼠标点击时还可以添加可选的细节项来描述正在寻找的键或鼠标按钮。对于鼠标按钮，1 为按钮 1，2 为按钮 2，3 为按钮 3。通常的鼠标设置为左边有按钮 1，右边有按钮 3，左撇子可以交换位置。对于键盘上的键，要么是键盘的字符（关于单字符键，如“a”或“*”键），要么是按键的名称。

下面是一些事件名称及描述，见表 8-18。

表 8-18 事件名称及描述

| 事件名称 | 描述 |
|------------------------------|--|
| <Button-1> | 鼠标左键单击 |
| <Button-2> | 鼠标中键单击 |
| <Button-3> | 鼠标右键单击 |
| <B1-Motion> | 左键移动 |
| <ButtonRelease-1> | 左键释放 |
| <Double-Button-1> | 双击左键 |
| <Shift-A> | 单击“Shift+A”组合键是事件发生，也可以将 Alt, Shift, Ctrl 和其他键组合使用 |
| <Shift_L> | 响应左侧的键为 Shift, Shift_L, F5 |
| <Triple-Button-1> | 三击鼠标左键为事件发生 |
| <Return>(F1,F2,F3,Delete...) | 按 Enter 键为事件发生，也可以将键盘上的任意键和一个时间绑定 |
| <KeyPress-H> | 用户按下了 H 键 |

2. 事件类型

Tkinter 整个事件类型的集合相当大，但其中常用的如表 8-19 所示。

表 8-19 事件类型及描述

| 事件类型 | 事件名称 | 描述 |
|------|---------------|--|
| 36 | Activate | 当组件的状态从未激活变成激活时触发该事件 |
| 4 | Button | 当用户单击鼠标按键时触发该事件。detail 部分指定具体按键： <Button-1> 鼠标左键，<Button-2> 鼠标中键，<Button-3> 鼠标右键， <Button-4> 滚轮上滚（Linux），<Button-5> 滚轮下滚 |
| 5 | ButtonRelease | 当用户释放鼠标按键时触发该事件 |
| 22 | Configure | 当组件的尺寸发生变化时触发该事件 |
| 37 | Deactivate | 当组件的状态从激活变成未激活时触发该事件 |
| 17 | Destroy | 当组件被销毁时触发该事件 |
| 7 | Enter | 当鼠标指针进入组件时触发该事件。注意：不是用户按下回车键 |
| 12 | Expose | 当窗口或者组件的某部分不再被覆盖时触发该事件 |
| 9 | FocusIn | 当组件获得焦点时触发该事件 |
| 10 | FocusOut | 当组件失去焦点时触发该事件 |
| 2 | KeyPress | 当用户按下键盘按键时触发该事件 |
| 3 | KeyRelease | 当用户按下键盘按键弹起时触发该事件 |
| 8 | Leave | 当鼠标指针离开组件时触发该事件。注意：不是用户按下回车键 |
| 19 | Map | 当组件被映射时触发该事件 |
| 6 | Motion | 当鼠标在组件内移动的整个过程中均触发该事件 |
| 38 | MouseWheel | 当鼠标滚动时触发该事件 |
| 18 | Unmap | 当一个组件被取消映射并不再可见时触发该事件。例如，当你使用组件方法“ <code>.grid_remove()</code> ”时，触发该事件 |
| 15 | Visibility | 当应用程序窗口的某个部分在屏幕上可见时就会发生 |

3. 事件修饰符

在事件序列中使用的修饰符名称及说明见表 8-20。

表 8-20 事件修饰符及说明

| 事件修饰符 | 说明 |
|---------|---|
| Alt | 当用户按住 Alt 键时为真 |
| Any | 修饰符推广事件类型。例如，事件模式“<Any-KeyPress>”适用于按任意键 |
| Control | 当用户按住 Ctrl 键时为真 |
| Double | 指定两个事件在时间上连续发生。例如，“<Double-Button-1>”描述鼠标 1 键双击 |
| Lock | 当用户按下换档锁“Shift+Lock”组合键时为真 |
| Shift | 当用户按住换档 Shift 键时为真 |
| Triple | 类似于 Double 事件，但指定三个事件快速连续发生 |

你可以使用短格式的事件形式：

<l>等同于<Button-l>，表示用户单击鼠标左键。

<h>等同于<KeyPress-H>，表示用户按下 H 键。



注意

在<...>短格式中，对于单字符按键不能使用<space>和<<>等。

4. 编写事件处理程序

处理程序被传递给一个事件对象“Event”，该对象描述发生了什么。处理程序可以是函数，也可以是方法。

下面是函数的定义：

```
def handlerName ( event ):
```

下面是方法的定义：

```
def handlerName ( self, event ):
```

Tkinter 事件传递给处理程序的事件对象的属性描述如表 8-21。下面属性中，有些总是被设置的，有些只是为特定类型的事件设置的。

表 8-21 Tkinter 事件传递给处理程序的事件对象的属性

| 属性 | 描述 |
|----------|---|
| .char | 如果事件与相关或为密钥产生一个规则的 ASCII 字符，则此字符串将设置为该字符 |
| .delta | 对于鼠标滚轮事件，此属性包含一个整数，其符号为正向上滚动，负向下滚动。 在 Windows 系统下，此值将为倍数“.delta 120”，如 120 表示向上滚动一步，-240 表示向下滚动两步。 在 macOS 系统下，它是 1 的倍数，所以 1 表示向上滚动一步，而-2 表示向下滚动两步。 有关 Linux 系统鼠标轮盘的支持，请参见按钮上的注释 |
| .height | 如果事件是 Configure，则此属性就设置为部件的新高度，单位为像素 |
| .keycode | 对于按键按下 KeyPress 事件或按键释放 KeyRelease 事件，此属性都被设置为标识按键的数字代码。然而，其并没有识别出按键是哪个字符产生的，因此使得 X 和 x 有相同的.keyCode 值 |
| .keysym | 事件涉及一个特殊键，此属性设置为键的字符串的名称。对于包含特殊按键的按键按下 KeyPress 或按键释放 KeyRelease 事件，此属性都设置为按键的字符串名称，如分页 PageUp 键为 Prior |



续表

| 属性 | 描述 |
|-------------|--|
| .keysym_num | 对于按键按下 KeyPress 或按键释放 KeyRelease 事件，“.keysym” 字段都设置为数字版本。对于产生单个字符的键，此字段设置为整数值的键的 ASCII 码 |
| .num | 如果事件与鼠标按钮有关，则此属性设置为按钮号（1,2 或 3）。对于 Linux 系统下的鼠标滚轮支持，绑定按钮 4 和按钮 5 事件；当鼠标滚轮向上滚动时此字段为 4，当向下滚动时为 5 |
| .serial | 在服务器每次处理客户机请求时都会递增的整数序列号，可以使用“.serial” 值来寻找事件的准确时间序列；值较低的事件发生得更快 |
| .state | 描述所有修饰键状态的整数。有关此值的解释，请参见下面的修饰符掩码表 |
| .time | 这个属性被设置为一个没有绝对意义的整数，但每毫秒都会增加。 这可以确定在应用程序中两次鼠标点击之间的时间长度 |
| .type | 描述事件类型的数字代码 |
| .widget | 始终设置为导致事件的部件。例如，如果事件是发生在画布上的鼠标单击，则此属性将是实际的 Canvas 画布控件 |
| .width | 如果事件是 Configure，则此属性将设置为部件的新宽度，单位为像素 |
| .x | 当事件发生时鼠标的 X 坐标，相对于部件的左上角 |
| .y | 当事件发生时鼠标的 Y 坐标，相对于部件的左上角 |
| .x_root | 鼠标在事件发生时位于屏幕上左上角的 X 坐标 |
| .y_root | 鼠标在事件发生时位于屏幕上左上角的 Y 坐标 |

使用掩码来测试“.state” 状态值，事件期间按下的修饰符键和按钮，见表 8-22。

表 8-22 事件期间按下的修饰符键和按钮

| 掩码 Mask | 修饰符 |
|---------|------------|
| 0x0010 | Num Lock 键 |
| 0x0080 | 右手 Alt 键 |
| 0x0100 | 鼠标按键 1 |
| 0x0200 | 鼠标按键 2 |
| 0x0400 | 鼠标按键 3 |

前面例子演示了如何将鼠标按钮在一块名为“w”的画布上画点，其中 paint() 就是事件处理程序，如下。

```
def paint( event ) :
    python_green = "#476042"
    x1, y1 = ( event.x - 1 ), ( event.y - 1 )
```

```
x2, y2 = ( event.x + 1 ), ( event.y + 1 )
w.create_oval( x1, y1, x2, y2, fill = python_green )
```

在调用此处理程序时，当前鼠标的位置为(event.x, event.y)。在利用 w.create_oval()方法绘制一个圆形时，以该位置为中心。

下面是绑定这个处理程序的代码。

```
w.bind( "<B1-Motion>", paint )    #绑定鼠标左键
```

Tkinter 的事件类型很丰富，它们为我们设计复杂的窗口图形提供了很多方法，在此就不一一介绍了。

8.10 Ttk控件

Tkinter 的主题控件 Ttk 是 Python 新版本中的标准库，无需再安装。

Tkinter 一直是 Python 的一个组成部分，它提供了一种与平台无关的窗口工具包。早期由于 Tkinter 包的功能有限，程序员做出的窗口界面单调古板，后来就推出了 tkinter.ttk 库（Tk 主题控件）。

Tk 主题控件带有 17 个小控件，其中 11 个已经存在于 Tkinter 中，即 Button，Checkbutton，Entry，Frame，Label，LabelFrame，Menubutton，PanedWindow，Radiobutton，Scale 和 Scrollbar，6 个新的窗口小控件是 Combobox，Notebook，Progressbar，Separator，Sizegrip 和 Treeview。

Ttk 的用法与 Tkinter 基本相同，但是 Ttk 不再支持有一些属性，如 Tkinter 中的“fg”和“bg”，它是通过 style 对象进行支持的。

例如，改变字体：

```
ttk.Style().configure(".", font=("仿宋", 25))
```

tkinter.ttk 库是 Tkinter 库的一部分，其能够增强 Tkinter 的界面设计，后面一起给大家介绍 Tkinter 程序设计。

tkinter.ttk 库的引入方式：

```
import tkinter as tk          #装载tkinter，用于Python 3
import tkinter.ttk as ttk     #导入Tkinter.ttk，用于Python 3
```


下面我们看一个 Tkinter 和 Ttk 混合编写的示例 8-21。

```
import tkinter as tk          #装载 tkinter, 用于 Python 3
import tkinter.ttk as ttk     #导入 Tkinter.ttk, 用于 Python 3
root =tk.Tk()                 #创建窗口对象
root.title(string = '主窗口标题') #设置窗口标题
frame = tk.Frame(root, width=200, height=100,bg='#ACACAC')
                                #创建 tk 的 Frame
frame.pack()                  #放置 Frame
frame2 = ttk.LabelFrame(root, text='选项', width=200, height=100)
                                #创建 ttk 的 LabelFrame
frame2.pack()                 #放置 Frame
root.mainloop()              #进入消息循环
```

程序运行结果见图 8-23。



图 8-23 示例程序运行结果

本节只介绍 Ttk 最常用的两个控件：ttk.Notebook 控件和 ttk.Treeview 控件。了解了这两个控件，就很容易掌握其他“ttk”控件了。

1. ttk.Notebook 控件

ttk.Notebook 控件类似于 Frame 控件，即通过点击顶部标签的选项卡选择不同容器。

构造函数：

```
w= ttk.Notebook(parent, option=value, ...)
```

其中，parent 为父容器，option 的属性关键字及说明见表 8-23。

表 8-23 Notebook 控件属性关键字

| 参数 | 说明 |
|-----------|--------------------------------|
| class | 控件类名，这是在创建控件时指定的，以后不能改变 |
| cursor | 光标形状 |
| height | 高度 |
| padding | 设置四个边的边距 |
| style | 风格 |
| takefocus | 在默认情况下，ttk.Notebook 控件将包含在焦点遍历 |
| width | 宽度 |

Notebook 控件的方法及说明见表 8-24。

表 8-24 Notebook 控件的方法

| 方法 | 说明 |
|--------------------------------|--|
| .add(child, **kw) | 增加一个新的选项卡 |
| .enable_traversal() | 选择选项卡的特定热键 |
| .forget(child) | 删除选项卡 |
| .hide(tabId) | 隐藏选项卡 |
| .index(tabId) | 返回对应的选项卡的数值。如果参数是字符串“end”，则该方法就返回标签的总数 |
| .insert(where, child, **kw) | 加入选项卡 |
| .select([tabId]) | 选择选项卡，并激活 |
| .tab(tabId, option=None, **kw) | 修改选项卡属性 |
| .tabs() | 此方法返回一个窗口名称的列表的子窗格中，从第一个到最后一个的顺序 |

其中，.add()方法和.tab()方法的选项卡有如下属性，见表 8-25。

表 8-25 Notebook 控件的选项卡属性

| 选项卡属性 | 说明 |
|-----------|--|
| compound | 如果同时提供 image 和 text 并要显示在选项卡中，则 compound 选项就是指定如何显示它们。使用值描述是文本在图像上的位置，取值为 tk.BOTTOM, tk.TOP, tk.LEFT, tk.RIGHT, tk.CENTER。例如，compound=tk.LEFT 为将文本放置在图片的左边 |
| padding | 四边边距，如 padding='0.1i' |
| sticky | 使用这个选项来指定面板内容定位的位置 |
| image | 图形图像显示在选项卡上 |
| text | 文本出现在标签上 |
| underline | 如果这个选项有一个负值 n，那么就会出现一个下画线的字符位置 n 下标签上的文本 |

我们来看示例 8-22。

```
import tkinter as tk                #装载 tkinter, 用于 Python 3
from tkinter import ttk            #装载 tkinter.ttk, 用于 Python 3
root =tk.Tk()                       #创建窗口对象
root.title(string = '主窗口标题')  #设置窗口标题
root.geometry('400x300+200+200')
frame2 = ttk.LabelFrame(root, text='ttk.LabelFrame', width=200, height=100)
                                     #创建 ttk 的 LabelFrame
frame2.pack(expand=1, fill="both") #放置 Frame
tabControl = ttk.Notebook(frame2)  #创建 Notebook
tab1 = ttk.Frame(tabControl)       #增加新选项卡
tabControl.add(tab1, text='信息窗') #把新选项卡增加到 Notebook
tab2 = ttk.Frame(tabControl)
tabControl.add(tab2, text='综合信息')
tab3 = ttk.Frame(tabControl)
tabControl.add(tab3, text='技术分析')
tab4 = ttk.Frame(tabControl)
tabControl.add(tab4, text='编写代码')
tab5 = ttk.Frame(tabControl)
tabControl.add(tab5, text='模拟回测')
tab6 = ttk.Frame(tabControl)
tabControl.add(tab6, text='双色球')
tab7 = ttk.Frame(tabControl)
tabControl.add(tab7, text='大乐透')
tabControl.pack(expand=1, fill="both")
tabControl.select(tab3)            #选择 tab3
root.mainloop()                   #进入消息循环
```

程序运行结果见图 8-24。

2. ttk.Treeview 控件

ttk.Treeview 控件可以做树型结构显示, 也可以当作表格来使用, 其支持水平滚动和垂直滚动。

构造函数:

```
w= ttk.Treeview(parent, option=value, ...)
```

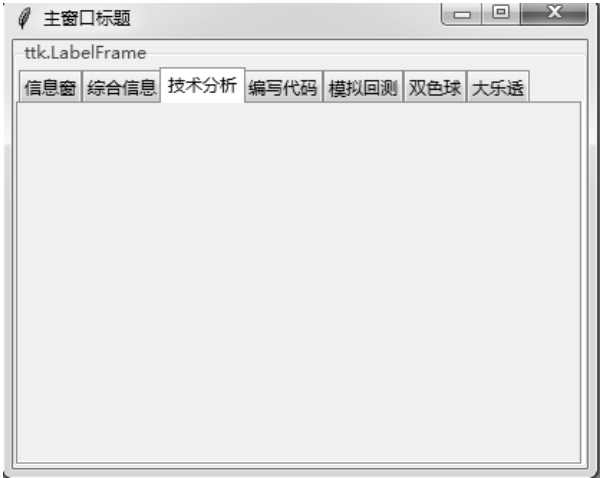


图 8-24 示例程序运行结果

其中，parent 为父容器，option 的属性关键字及说明见表 8-26。

表 8-26 Treeview 控件的属性关键字

| 参数 | 说明 |
|----------------|------------------------------------|
| columns | 列标识符的列表，指定列的数量和名字 |
| displaycolumns | 列标识符的列表（象征性或整数指数）指定显示哪些数据列及它们出现的顺序 |
| height | 指定的行数应清晰可见。注意：要求宽度根据列宽之和确定 |
| padding | 指定控件的内部填充 |
| selectmode | 控制内置类绑定管理选择 |
| show | 下面的列表包含的值，指定哪些元素树的显示 |

Treeview 项目的属性及说明见表 8-27。

表 8-27 Treeview 项目的属性

| 属性 | 说明 |
|--------|---------------------------------|
| text | 项目的文字 |
| image | Tk 图像，显示左边的标签 |
| values | 与项目相关联的值的列表 |
| open | 值“True”或“Flase”指示项的子项目是否应该显示或隐藏 |
| tags | 与这个项目相关的标签列表 |

Treeview 的 Tag 标签选项及说明见表 8-28。

表 8-28 Tag 标签选项

| 选项 | 说明 |
|------------|-----------------|
| foreground | 指定文本前景颜色 |
| background | 指定单元格背景颜色或项目 |
| font | 在指定时所使用的字体绘制文本 |
| image | 指定项目形象，防止图像选项为空 |

ttk.Treeview 控件的虚拟事件及说明见表 8-29。

表 8-29 Treeview 控件的虚拟事件

| 虚拟事件 | 说明 |
|--------------------|-----------------|
| <<TreeviewSelect>> | 只要选择发生变化 |
| <<TreeviewOpen>> | 在“open=True”之前 |
| <<TreeviewClose>> | 在“open=False”之后 |

Treeview.focus()方法和 Treeview.selection()方法可以作用于要变化的项目。

ttk.Treeview 类的方法及说明见表 8-30。

表 8-30 Treeview 方法

| 方法 | 说明 |
|------------------------------------|--|
| bbox(item, column=None) | 返回边界框（相对于 Treeview 控件的窗口）指定条目的形式（x,y, 宽度，高度） |
| get_children(item=None) | 返回属于它们的子项列表 |
| set_children(item, *newchildren) | 替换项与 newchildren 的子项 |
| column(column, option=None, **kw) | 查询或修改选项指定列 |
| delete(*items) | 删除所有指定的项目和它们的后代 |
| detach(*items) | 从树上拆开所有指定的项目 |
| exists(item) | 如果指定的项目在树中存在，则返回 True |
| focus(item=None) | 如果指定项目就设置为焦点项目。否则，返回当前的焦点项目 |
| heading(column, option=None, **kw) | 查询或修改标题选项指定的列 |
| identify(component, x, y) | 返回指定（x,y）坐标上的控制件。如果没有这样的控件出现，就返回空字符串 |
| identify_row(y) | 返回项的 ID 项在 y 的位置 |
| identify_column(x) | 返回单元格的数据列标识符在 x 的位置 |

续表

| 方法 | 说明 |
|--|---|
| <code>identify_region(x, y)</code> | 返回标题或单元或区域数据之一 |
| <code>identify_element(x, y)</code> | 返回元素的 <code>x</code> , <code>y</code> 位置 |
| <code>index(item)</code> | 返回项的整数索引在其父母的子项 |
| <code>insert(parent, index, iid=None, **kw)</code> | 创建一个新的条目并返回新创建的条目的标识符 |
| <code>item(item, option=None, **kw)</code> | 查询或修改项目指定的选项 |
| <code>move(item, parent, index)</code> | 移动项目位置指数在父母的孩子 |
| <code>next(item)</code> | 返回条目的标识符的下一个兄弟姐妹, 或""如果项目是最后一个子项的父母 |
| <code>parent(item)</code> | 返回的父项的 ID, 或者""如果项目是在层次结构的顶层 |
| <code>prev(item)</code> | 返回条目的标识符之前的兄弟姐妹, 或""如果项目是父母的第一个孩子 |
| <code>reattach(item, parent, index)</code> | 别名为 <code>Treeview.move()</code> |
| <code>see(item)</code> | 确保项目是可见的 |
| <code>selection(selop=None, items=None)</code> | 如果没有指定 <code>selop</code> 参数, 就返回选定的项目。否则, 就根据 <code>selop</code> 返回项目的方法 |
| <code>selection_set(*items)</code> | 选择项目 |
| <code>selection_add(*items)</code> | 将条目添加到选择 |
| <code>selection_remove(*items)</code> | 从选择删除项中删除选择的项目 |
| <code>selection_toggle(*items)</code> | 切换中每一项物品的选择状态 |
| <code>set(item, column=None, value=None)</code> | 如果有一个参数, 就返回一个指定的列/值对字典条目。如果有两个参数, 就返回指定列的当前值。如果有三个参数, 就在给定的项目中设置为指定的值 |
| <code>tag_bind(tagname, sequence=None, callback=None)</code> | 此方法将绑定到已标记的所有项目参数所指定的事件处理程序 |
| <code>tag_configure(tagname, option=None, **kw)</code> | 查询或修改 <code>tagname</code> 指定的选项 |
| <code>tag_has(tagname, item=None)</code> | 如果指定项, 就返回 1 或 0, 取决于指定的项是否给定 <code>tagname</code> 。否则, 返回一个列表的所有项目指定的标签 |
| <code>xview(*args)</code> | 查询或修改 <code>Treeview</code> 的水平位置 |
| <code>yview(*args)</code> | 查询或修改 <code>Treeview</code> 的垂直位置 |

ttk.Treeview 控件的功能非常强大, 这里就不一一介绍了。

下面见示例 8-23。

```

import tkinter as tk                    #装载tkinter库, 用于Python 3
from tkinter import ttk                #装载tkinter.ttk库, 用于Python 3
root =tk.Tk()                          #创建窗口对象
root.title(string = 'ttk.Treeview 演示')#设置窗口标题
root.geometry('600x500+200+200')
tree=ttk.Treeview(root,height=9)
myid=tree.insert("",0,"指标",text="指标",values=("1"),open=True)
                                     #""表示父节点是根
myidx1=tree.insert(myid,0,"技术指标",text="技术指标",values=("2"))
                                     #text 表示显示出的文本, values 是隐藏的值
myidx2=tree.insert(myid,1,"交易系统",text="交易系统",values=("3"))
myidx3=tree.insert(myid,2,"条件选股",text="条件选股",values=("4"))
myidx4=tree.insert(myid,3,"五彩K线",text="五彩K线",values=("5"))
myidy=tree.insert("",1,"工具",text="工具",values=("6"),open=True)
myidy1=tree.insert(myidy,0,"系统设置",text="系统设置",values=("7"))
myidy1=tree.insert(myidy,0,"指标排序",text="指标排序",values=("8"))
tree.pack(fill=tk.X)
tree.selection_set("指标")
tree2 = ttk.Treeview(root, columns=('col1','col2','col3'),show="headings")
tree2.column('col1', width=100, anchor='center')
tree2.column('col2', width=100, anchor='center')
tree2.column('col3', width=100, anchor='center')
tree2.heading('col1', text='股票代码')
tree2.heading('col2', text='日期')
tree2.heading('col3', text='时间')
def onDBCclick(event):
    item = tree2.selection()[0]
    print( "you clicked on ", tree2.item(item, "values"))
for i in range(10):
    tree2.insert('',i,values=('6000'+str(i),'2019 年 9 月 3 日','9:31'))
tree2.bind("<Double-1>", onDBCclick)
tree2.pack(fill=tk.X)
root.mainloop()

```

程序运行结果见图 8-25。



图 8-25 示例程序运行结果

8.11 Tix控件

Tkinter 的扩展控件 Tix 是 Python 新版本中的标准库，无需再安装。

Tix 库提供了标准 Tk 库缺失的常用小控件，如 HList 控件、ComboBox 控件、Control 控件和各种可滚动的小控件。由于标准的 Tk 控件过于原始，因此借助 Tix 超级控件可以设计出更加美观的应用程序。tkinter.tix 包括 40 多个控件。

我们使用这些新的控件可以在应用程序中引入新的交互技术，创建更有用和更直观的用户界面，还可以通过选择最合适的控件来设计应用程序，以满足应用程序和用户的特殊需求。

1. Tix 的使用

在 Python 3 中，Tix 已被重命名为 tkinter.tix。

前面我们学习了 Ttk，Ttk 是建立在 Tkinter 基础上的，因此除了 Tkinter 的基本控件外，我们还可以在程序中灵活使用 Ttk 所提供的控件。

也就是说，Ttk 的控件完全可以适用在 Tkinter 的窗口和容器中，而 Tkinter.Tk() 方法的实例对象“root”管理整个 Tkinter 和 Ttk 控件的运行。

```
import tkinter as tk
```

```
#导入 Tkinter
```



```
import tkinter.ttk as ttk          #导入 Tkinter.ttk

root=tk.Tk()                       #Tkinter.Tk() 顶层调用
```

但是在 Tix 模块类中，将 Tkinter 模块中的类子类化，因此 Tix 是建立在 Tkinter 模块基础上的，故 Tkinter.Tk()方法的实例对象“root”无法管理 Tix 控件。

由于 Tix 模块中的 Tix.Tk()类继承于 tkinter.Tk 类，因此，它除了能管理 Tix 控件外，还能管理 Tkinter 控件和 Ttk 控件。

如果要将 Tix 控件和 Tkinter 控件一起使用，只需要导入一个模块即可。一般情况下，可以只导入 Tix，用 Tix.Tk()替换 Tkinter.Tk()的顶层调用并负责管理 Tix 和 Tkinter 控件的运行。因此，需要用 Tix.Tk()方法的实例对象“root”管理 Tix 控件、Tkinter 控件和 Ttk 控件的运行。

下面分别给出导入 Tkinter 库、Ttk 库、Tix 库的代码。

```
import tkinter as tk              #导入 Tkinter
import tkinter.ttk as ttk        #导入 Tkinter.ttk
import tkinter.tix as tix        #导入 Tkinter.tix
from tkinter.constants import *  #导入 Tkinter 常量
root=tix.Tk()                    #Tkinter.tix.Tk() 顶层调用
```

在程序中，可以同时使用 Tkinter（别名“tk”）库控件、Ttk（别名“ttk”）库控件、Tix（别名“tix”）库控件等，但是一些方法被 Tix 更高级的方法替代或增强了。例如，tk.Label 的几何管理除了能够使用 Pack 方法、Grid 方法和 Place 方法外，还增加了新的高级管理方法——Form 方法。

通过上面的方式导入模块，在程序中我们可以使用 Tkinter 常量中的 W，tk.W 和 tix.W。

tix.Tk()对象“root”拥有 tk.Tk()对象的全部属性和方法，也就是说 tix.Tk()对象可以使用 Tkinter 中 tk.Tk()对象的全部方法和属性。在上面的程序段中用 root= tix.Tk()替代 root=tk.Tk()，其他内容不用改变，其运行结果相同。

Tix 控件与 Ttk 控件一样，对于外观属性，如字体等不再使用 Tkinter 的属性关键字 font 来设置。Tix 控件使用属性 options 通过字符串来设置。

外观属性 options 的字符串描述格式为“控件.属性 参数值...”。

例如，options='listbox.height 4 label.padY 5 label.width 10 label.anchor ne'。

2. Form 表格几何管理器

Tix 提供了新的表格几何管理器——Form，该管理器适用于所有的 Tkinter 控件布局的规则。

Form 表格几何管理器用法如下：

```
w.form(top=0, left=0, right=None, bottom=None)
```

在父容器控件中放置（显示）控件 `w`，其四边大小和位置由参数控制。其中，`top` 和 `left` 相当于控件“`w`”在父控件的相对坐标 (x,y) ，单位是像素。

当 `right` 和 `bottom` 的值为 `None` 时，控件 `w` 为自动宽度。

当 `right` 和 `bottom` 的值为 `-1` 时，控件 `w` 的右边到父控件的右边框，或者控件 `w` 的下边到父控件的下边框。

当 `top`，`left`，`right` 和 `bottom` 为正整数时，表示父控件的绝对坐标。

当 `top`，`left`，`right` 和 `bottom` 使用“`%n`”格式的字符串时，表示父控件的相对坐标。例如，`left="%60"` 表示 `w` 左边的相对位置坐标在父容器控件宽度的 60% 位置。

在 `top`，`left`，`right`，`bottom` 中，其中一个可以是其他使用 Form 布局的控件 `w2`。例如，`top=w2` 表示 `w` 的 `top` 等于 `w2` 的 `bottom`。

```
w.forget()
```

`w.forget()` 方法可以删除掉控件 `w` 的显示，但控件 `w` 对象还存在。

下面看一个示例 8-24。

```
import tkinter as tk          #导入 Tkinter
import tkinter.ttk as ttk     #导入 Tkinter.ttk
import tkinter.tix as tix     #导入 Tkinter.tix
from tkinter.constants import * #导入 Tkinter 常量, W, tk.W 和 tix.W 都可用
root = tix.Tk()               #创建 tix.Tk() 主窗口
width=300                      #把窗口宽度 300 (单位:像素) 赋值给变量 width
height=200                    #把窗口高度 300 (单位:像素) 赋值给变量 height
x,y=150,250                   #给屏幕坐标 (x,y) 赋值 (100,200)
root.geometry('{}x{}+{}+{}'.format(width,height, x, y))
                                #改变窗口的位置和大小
root.title("Tix 的 Form 布局演示")
la=tk.Label(root, text='aaa', relief=tix.SUNKEN, bd=1) #创建 tk.Label
la.form(top=50, left=50, right=None, bottom=None)      #用 Form 方法显示
```

```

lb=ttk.Label(root, text='bbb')      #创建 ttk.Label
lb.place(x=50,y=100)                #用 Place 方法显示
lc=tix.Label(root, text='ccc',relief=tix.SUNKEN, bd=1) #创建 tix.Label
lc.form(top='%30',left="%50")       #用 Form 方法显示
ld=tk.Label(root, text='ddd',relief=tix.SUNKEN, bd=1) #创建 tk.Label
ld.form(left="%50", top=lc,bottom=-1) #用 Form 方法显示
le=tk.Label(root, text='eee',relief=tix.SUNKEN, bd=1) #创建 tk.Label
le.form(left=lc, top="%30",right=-1) #用 Form 方法显示
root.mainloop()                    #开启 tk 主循环

```

程序运行结果见图 8-26。



图 8-26 示例程序运行结果

从上面程序代码和运行结果中可知，使用 `tix.Tk()` 方法建立主窗口后，使得 Tkinter 控件、Ttk 控件和 Tix 的控件都具有了 Form 方法。Form 方法类似于 Place 方法，但是使用 Form 方法的控件都具有了控件的“锚定”特性，这可以成为其他控件布局定位的参考。

由于 Tix 的很多控件与 Tkinter 控件的用法类似，因此下面只介绍 Tix 特有的几个控件。

3. tix.Balloon 气球窗口控件

`tix.Balloon` 气球窗口控件可以为其他控件提供帮助。当用户将光标移动到已绑定了该控件的窗口控件中时，屏幕上就会显示一个带有描述性消息的小型弹出式窗口。

使用 `tix.Balloon` 气球窗口控件的构造语法如下：

```
balloon = tix.Balloon (master, statusbar=None, options="")
```

其中, `master` 为父控件。`statusbar` 是此控件绑定的状态条控件。`options` 用于配置控件。

除了上面的属性外, `tix.Balloon` 控件还有一些方法可以使用。

`.bind_widget()`方法绑定控件的格式如下:

```
balloon.bind_widget(widget, balloonmsg='', statusmsg=None)
```

其中, `widget` 是要绑定的控件, 也就是说在鼠标移动到控件上方时会出现需要提示的信息。

`balloonmsg` 是需要提示的信息。

`statusmsg` 是需要在状态栏显示的信息。

`.unbind_widget()`方法解绑控件的格式如下:

```
balloon.unbind_widget(widget)
```

其中, `widget` 是要解绑的控件。

下面看一个 `tix.Balloon` 气球窗口控件的示例 8-25。

```
import tkinter.tix as Tix          #导入 Tkinter.tix
from tkinter.constants import *
TCL_ALL_EVENTS = 0
def RunSample (root):
    balloon = DemoBalloon(root)
    balloon.mainloop()
    balloon.destroy()
class DemoBalloon:
    def __init__(self, w):
        self.root = w
        self.exit = -1
        z = w.winfo_toplevel()
        z.wm_protocol("WM_DELETE_WINDOW", lambda self=self: self.quitcmd())
        z.title('Tix.Balloon 演示')    #在 Tkinter 中设置窗口标题方法
        status = Tix.Label(w, width=40, relief=Tix.SUNKEN, bd=1)
        status.pack(side=Tix.BOTTOM, fill=Tix.Y, padx=2, pady=1)
        #建立两个 Tix 按钮
        button1 = Tix.Button(w, text='关闭窗口',
                             command=self.quitcmd)
        button2 = Tix.Button(w, text='按钮自毁')
```

```

button2['command'] = lambda w=button2: w.destroy()
button1.pack(side=Tix.TOP, expand=1)
button2.pack(side=Tix.TOP, expand=1)
#建立 tixballoon
b = Tix.Balloon(w, statusbar=status)
b.bind_widget(button1, balloonmsg='关闭这个窗口',
               statusmsg='按下这个按钮,关闭窗口。')

b.bind_widget(button2, balloonmsg='删除这个按钮',
               statusmsg='按下这个按钮,删除这个按钮。')

def quitcmd (self):
    self.exit = 0
def mainloop(self):
    foundEvent = 1
    while self.exit < 0 and foundEvent > 0:
        foundEvent = self.root.tk.dooneevent(TCL_ALL_EVENTS)
def destroy (self):
    self.root.destroy()
if __name__ == '__main__':
    root = Tix.Tk()
    RunSample(root)

```

程序运行结果见图 8-27。

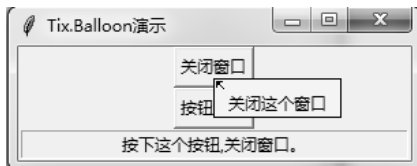


图 8-27 示例程序运行结果

4. tix.ButtonBox 按钮框控件

tix.ButtonBox 按钮框中可以添加一些按钮，如通常用的“Ok”和“Cancel”等。

按钮框中添加的所有按钮的宽度值要相同，另外按钮提示文本、下画线、命令和宽度选项都是 tix.Button 按钮控件的标准选项。

tix.ButtonBox 控件的构造语法如下：

```
btnbox = tix.ButtonBox (master, orientation=HORIZONTAL)
```

其中，`master` 为父控件。`orientation` 是控件排列按钮的方向，默认水平为 `HORIZONTAL`。

除了上面的属性外，`tix.ButtonBox` 控件还有一些方法可以使用。

`.add()`方法增加按钮的语法如下：

```
btnbox.add(name, option=value, ...)
```

其中，`name` 是按钮的名称。`option` 是 `Tix.Button` 按钮控件的一些属性参数，如 `text`, `underline`, `width`, `command` 等。

`.invoke()`方法调用按钮的 `command` 回调程序，其语法如下：

```
btnbox.invoke(name)
```

其中，`name` 是 `tix.ButtonBox` 控件添加的按钮名称。

调用名字为“`name`”按钮的“`command`”回调函数或方法，并返回该函数返回的内容，这类似于程序模拟按钮动作。如果按钮被禁用或没有回调，则无效。

不像 `tk.messagebox` 中的一些控件那样会返回参数值，`tix.ButtonBox` 只能使用 `command` 参数写回调函数。不过，使用 `Tix` 并不会影响我们对 `Tkinter` 原始控件的使用。

下面看一个有关 `tix.ButtonBox` 控件的演示程序。当点击“Quit”按钮时，会弹出一个“`tk.messagebox`”对话框，再确认一次退出操作。见示例 8-26。

```
import tkinter as tk                #导入 Tkinter
from tkinter import messagebox, filedialog, simpledialog, colorchooser
import tkinter.tix as tix          #导入 Tkinter.tix
#移动窗口到屏幕坐标 x,y
def setPlace(window,x, y,w=0,h=0):
    if (w==0 or h==0):
        w = window.winfo_width()    #获取窗口宽度（单位：像素）
        h = window.winfo_height()    #获取窗口高度（单位：像素）
        window.geometry('{}x{}+{}+{}'.format(w, h, x, y))
def RunSample(w):
    top = tix.Label(w, padx=20, pady=10, bd=1, relief=tix.RAISED,
                    anchor=tix.CENTER, text='这个对话框演示 tixButtonBox 的使用,\n
默认 orientation= tix.HORIZONTAL')
    box = tix.ButtonBox(w)
    def quit():
```

```

ans=messagebox.askyesno('提示', '要程序结束吗?')
                                #确定/取消, 返回值 True/False

if ans==True:
    w.destroy()
box.add('ok', text='OK', underline=0, width=5,
        command=lambda w=w: w.destroy())
box.add('quit', text='Quit', underline=0, width=5,
        command=quit)
box.pack(side=tix.BOTTOM, fill=tix.X)
top.pack(side=tix.TOP, fill=tix.BOTH, expand=1)
def RunSample2(w):
    top = tix.Label(w, padx=20, pady=10, bd=1, relief=tix.RAISED,
                    anchor=tix.CENTER, text='这个对话框演示 tix.ButtonBox 的使用,\n
使用 orientation= tix.VERTICAL')
    box = tix.ButtonBox(w, orientation=tix.VERTICAL)
    box.add('ok', text='确定', width=5,
            command=lambda w=w: w.destroy())
    box.add('close', text='取消', width=5,
            command=lambda w=w: w.destroy())
    box.pack(side=tix.BOTTOM, fill=tix.X)
    top.pack(side=tix.TOP, fill=tix.BOTH, expand=1)
if __name__ == '__main__':
    root = tix.Tk()
    RunSample(root)
    top=tix.Toplevel(root)
    RunSample2(top)
    root.update()
    root.title('Tix.ButtonBox 演示') #在 Tkinter 中设置窗口标题的方法
    setPlace(root,100,100)
    top.title('Tix.ButtonBox 演示') #在 Tkinter 中设置窗口标题的方法
    setPlace(top,100,300)
    root.mainloop()

```

程序运行演示图如 8-28 所示。

5. tix.ComboBox 组合框控件

tix.ComboBox 组合框控件类似于 MS Windows 中的组合框控件, 用户可以通过键入条目子组件或从列表框子选择组件中来选择一个选项。

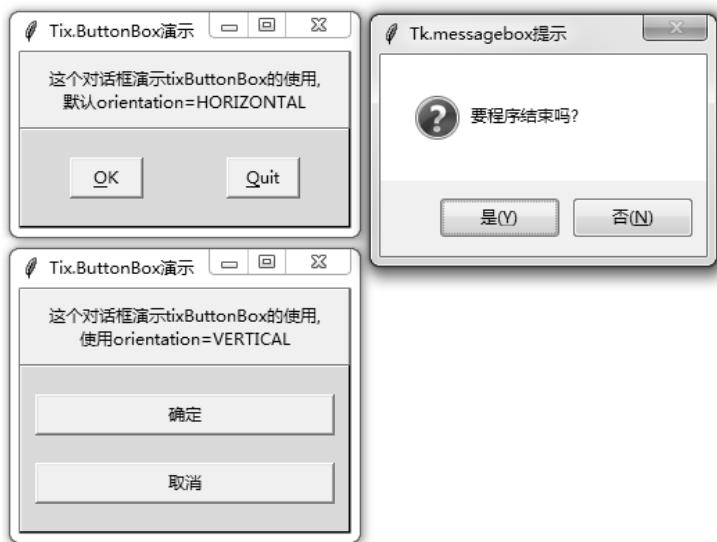


图 8-28 示例程序运行结果

tix.ComboBox 控件的构造语法如下：

```
cbox = tix.ComboBox (master, label, dropdown=1, editable=0, command=None,
                    fancy=0, variable=None, options='')
```

master 为父部件。

label 是要显示的标签。

dropdown 表示是否允许下拉，默认为 1 是允许下拉菜单方式显示。

editable 表示是否允许编辑，默认为 0 是不允许编辑。

command 为设置回调函数。

variable 为设置一个变量。

fancy 为在可编辑状态下设置一个删除和选择按钮，默认为 0。

options 为设置控件的外观式样等。

除了上面的属性外，tix.ComboBox 组合框控件还有一些方法可以使用。

.insert()为插入字符串，语法如下：

```
cbox.insert(index, str)
```

其中，index 是索引，如最后位置为 tix.END。str 是插入的字符串。

.pick()方法显示 index 位置的字符串，语法如下：


```
cbox.pick(index)
```

此方法使显示的第一个值为 `index` 索引的字符串。

`.add_history()`方法在最前面增加字符串，语法如下：

```
cbox.add_history( str)
```

把参数“`str`”字符串插入到选择值列表的最前面。

`.append_history()`方法在最后面增加字符串，语法如下：

```
cbox.append_history(str)
```

把参数“`str`”字符串插入到选择值列表的最后面。

`.set_silent()`方法设置默认值，语法如下：

```
cbox.set_silent(str)
```

把参数“`str`”设置为默认值。

下面看示例 8-27。

```
import tkinter.tix as tix      #导入 Tkinter.tix
def RunSample(w):
    global demo_month, demo_year
    top = tix.Frame(w, bd=1, relief=tix.RAISED)
    demo_month = tix.StringVar()
    demo_year = tix.StringVar()
    a = tix.ComboBox(top, label="月: ", fancy=1, editable=1, tick=0,
        command=select_month, variable=demo_month,
        options='listbox.height 6 label.width 6 label.anchor e')
    b = tix.ComboBox(top, label="年: ", dropdown=0, fancy=1,
        command=select_year, editable=1, variable=demo_year,
        options='listbox.height 4 label.padY 5 label.width 6 label.anchor ne')

    a.pack(side=tix.TOP, anchor=tix.W)
    b.pack(side=tix.TOP, anchor=tix.W)
    a.insert(tix.END, 'January')
    a.insert(tix.END, 'February')
    a.insert(tix.END, 'March')
    a.insert(tix.END, 'April')
    a.insert(tix.END, 'May')
```

```

a.insert(tix.END, 'June')
a.insert(tix.END, 'July')
a.insert(tix.END, 'August')
a.insert(tix.END, 'September')
a.insert(tix.END, 'October')
a.insert(tix.END, 'November')
a.insert(tix.END, 'December')
for y in range(2010,2020):
    b.insert(tix.END, '%d'%y)
a.set_silent('May')
b.set_silent('2019')
box = tix.ButtonBox(w, orientation=tix.HORIZONTAL)
box.add('ok', text='确定', underline=0, width=6,
        command=lambda w=w: ok_command(w))
box.add('cancel', text='取消', underline=0, width=6,
        command=lambda w=w: w.destroy())
box.pack(side=tix.BOTTOM, fill=tix.X)
top.pack(side=tix.TOP, fill=tix.BOTH, expand=1)
def select_month(event=None):
    #tixDemo:Status "Month = %s" % demo_month.get()
    pass

def select_year(event=None):
    #tixDemo:Status "Year = %s" % demo_year.get()
    pass
def ok_command(w):
    #tixDemo:Status "Month = %s, Year= %s" % (demo_month.get(), demo_
year.get())
    w.destroy()
if __name__ == '__main__':
    root = tix.Tk()
    root.title('tix.ComboBox 演示') #在 Tkinter 中设置窗口标题的方法
    RunSample(root)
    root.mainloop()

```

程序运行结果见图 8-29。



图 8-29 示例程序运行结果

6. tix.Control 控制窗口控件

tix.Control 控制窗口控件也被称为 SpinBox 窗口控件,用户可以通过按上下两个箭头按钮或直接输入数值来调整数值。新值将根据用户定义的上限和下限进行检查。

tix.Control 控制窗口控件的构造语法如下:

```
spinbox = tix.Control (master, label, integer=0, min=None, max=None,
                      step=None, variable=None, options='')
```

master 为父控件。

label 为要显示的标签。

integer 的默认值为 0, 当值为 1 时, 表示是整型数据。

min, max, step 一起使用, 其中 min 是数据的最小值, max 为最大值, step 为变动步长。

variable 可以设置一个变量。

options 可以设置部件的外观式样等。

除了上面的属性外, tix.Control 控制窗口控件还有一些方法可以使用。

.invoke()方法模拟控件的动作, 语法如下:

```
spinbox.invoke()
```

.update()方法更新控件的值, 语法如下:

```
spinbox.update()
```

.decrement()方法显示上一个数据, 语法如下:

```
spinbox.decrement()
```

.increment()方法显示下一个数据，语法如下：

```
spinbox.increment()
```

下面看一个 tix.Control 控制窗口控件的演示代码，见示例 8-28。

```
import tkinter as tk          #导入 Tkinter
import tkinter.tix as tix     #导入 Tkinter.tix
from tkinter.constants import *
TCL_ALL_EVENTS = 0
global maker_m
maker_m = []
maker_m.append('January')
maker_m.append('February')
maker_m.append('March')
maker_m.append('April')
maker_m.append('May')
maker_m.append('June')
maker_m.append('July')
maker_m.append('August')
maker_m.append('September')
maker_m.append('October')
maker_m.append('November')
maker_m.append('December')
def adjust_maker(w, inc):
    global maker_m,demo_m
    i = maker_m.index(demo_m.get())
    i = i + inc
    if i >= len(maker_m):
        i= 0
    elif i < 0:
        i=len(maker_m) - 1
    demo_m.set(maker_m[i])
def validate_maker(w):
    global maker_m,demo_m
    i=0
    try:
```

```

        i=maker_m.index(demo_m.get())
    except ValueError:
        return maker_m[0]
    return maker_m[i]
class RunSample:
    def __init__(self, w):
        self.root = w
        self.exit = -1
        global demo_m, demo_d, demo_year
        global maker_m
        demo_m = tix.StringVar()
        demo_d = tix.IntVar()
        demo_year = tix.IntVar()
        demo_d.set(1)
        demo_m.set('May')
        demo_year.set(2019)

        top = tix.Frame(w, bd=1, relief=tix.RAISED)
        a = tix.Control(top, label='年: ', integer=1,
                        variable=demo_year, min=2010, max=2030,
                        options='entry.width 20 label.width 10 label.anchor e')
        b = tix.Control(top, label='月: ', value='May',
                        variable=demo_m,
                        options='entry.width 20 label.width 10 label.anchor e')
        c = tix.Control(top, label='日: ', integer=1,
                        min='1', max='31', step=1,
                        variable=demo_d,
                        options='entry.width 20 label.width 10 label.anchor e')
        b['incrcmd'] = lambda w=b: adjust_maker(w, 1)
        b['decrcmd'] = lambda w=b: adjust_maker(w, -1)
        b['validatecmd'] = lambda w=b: validate_maker(w)
        demo_m.set('May')
        a.pack(side=tix.TOP, anchor=tix.W)
        b.pack(side=tix.TOP, anchor=tix.W)
        c.pack(side=tix.TOP, anchor=tix.W)
        box = tix.ButtonBox(w, orientation=tix.HORIZONTAL)
        box.add('ok', text='确定', underline=0, width=6,

```

```

        command=self.okcmd)
    box.add('cancel', text='取消', underline=0, width=6,
            command=self.quitcmd)
    box.pack(side=tix.BOTTOM, fill=tix.X)
    top.pack(side=tix.TOP, fill=tix.BOTH, expand=1)

    a.decrement()    #显示上一条数据
    b.update()        #刷新数据
    c.increment()     #显示下一条数据
def okcmd (self):
    self.quitcmd()
def quitcmd (self):
    self.exit = 0
    self.destroy()
def mainloop(self):
    while self.exit < 0:
        self.root.tk.dooneevent(TCL_ALL_EVENTS)

def destroy (self):
    self.root.destroy()
if __name__ == '__main__':
    root = tix.Tk()
    root.title('tix.Control 演示') #在 Tkinter 中设置窗口标题的方法
    RunSample(root)
    root.mainloop()

```

程序运行结果见图 8-30。



图 8-30 示例程序运行结果

由于 Tkinter 模块的功能非常多，因此本书只介绍其中一些典型的方法，这些方法在搭建量化平台中都可以用到。

9

第 9 章

小白量化投资分析平台

小白量化投资分析平台是单机量化分析系统。它集成了股票数据分析、K 线图显示、仿大智慧/通达信自定义指标编写和显示及实现选股和回测功能，同时可以根据用户需求进行扩展，如深度学习分析等。读者可以根据本书前面所学知识，搭建一个类似通达信界面的量化分析平台。

小白量化投资分析平台是利用 Python 3 编写的，因此可以跨平台使用，如可以在 Windows 7/8/10 的 32 位或 64 位操作系统上使用，也可以在 macOS 系统和 Linux 系统上使用。这个系统是一个开源的，用户可以看到全部源代码，这样既可以学习编写量化软件的思路，又可以根据自己的需要进行改进升级。

小白量化投资分析平台使用 NumPy 库、Pandas 库、Matplotlib 库进行数据统计分析，并且依赖 Tkinter 库搭建了软件的图形应用界面。

9.1 平台整体功能的划分

我们采用模块化思想把小白量化投资分析平台从整体上划分为功能相对独立的五个模块，分别为数据获取、数据研究、策略研究、历史回测和策略执行。



1. 数据获取

数据获取可以分为 API 数据获取、网络爬虫获取和文件获取三种方式，每种方式都可以开发出独立的模块。

2. 数据研究

数据研究可分为以下六种方式：

- (1) 根据股价和成交量来分析价格的变化趋势。
- (2) 根据股票基本数据、产业行业数据、财政政策等来分析选择股票或预测股票的价格趋势。
- (3) 股票价格和成交量指标的分析，主要通过数学公式变换和指标图形研究股票价格的变化规律。
- (4) 理论研究，通过对周期理论、波浪理论、分形理论、周易阴阳理论等的研究预测股票的价格趋势。
- (5) 量化分析，即利用因子模型、多维分析来预测股票的价格趋势。
- (6) 基于深度学习来预测股票的价格趋势。

3. 策略研究

通常，仅仅通过进行技术分析还不能保证资金池的收益最大化，因此还需要结合策略研究。如果把证券交易看作战争，那么数据研究就属于战术层面的研究，而策略研究就属于战略层面的研究。策略研究的重点是使风险最小化，同时使收益最大化。战略研究的目的是能赢，而不是赢多少。

4. 历史回测

历史回测是使用历史行情数据进行回测分析来检验操作策略是否有效，以便对策略参数进行调整。

5. 策略执行

由于策略执行涉及自动交易问题，因此其基本上为纯计算机技术问题。策略执行一般通过两种方式来实现：API 的接口和外挂方式。

以上是小白量化投资分析平台整体软件功能的划分，接下来主要介绍每个功能如何实现。

9.2 全局变量HP_global

根据 Python 语言的特点，软件总体设计可以划分为以下几个部分。

1. 定义全局变量

涉及软件每个模块都需要使用的变量，就将其放置到全局变量模块中，全局变量模块名为 HP_global.py。变量主要有以下几类：

(1) 软件本身信息。例如，软件名称标题、开发者信息、版本号、图标、最后构建日期等。

(2) 系统环境信息。例如，操作系统平台及版本号、Python 版本号、模块信息版本号等。

(3) 用户信息。例如，用户名及密码、是否登录成功信息，以及其他有关用户的信息，如性别、年龄等，这些都可以根据软件需要来设置。

(4) 软件特有的变量。例如，颜色表字典、鼠标形状列表等。为了方便用户使用，这些预先作了设置。

(5) 数据接口设置变量。由于数据接口很多，因此是选择 QUANTAXIS 数据（简称 qa 数据）还是选择 Tushare 数据（简称 ts 数据）需要提前设置，因为平台里的算法是根据设置来进行数据格式变换的。这涉及公式系统、绘图系统和回测系统等方面。

(6) 平台框架数据类型和个性化设置。例如，分析的数据类型，如股票、期货、彩票、数字币等，它们的分析存在细微差别。

(7) 绘图相关信息。例如，默认分析指标、指标默认参数等。

由于 Python 区分大小写，我们这里都采用小写作为全局变量。下面给出主要代码：

```
#运行系统环境设置
#os=1 windows,=2 linux, =3 mac os
global os           #操作系统
global pyver        #Python 版本
#软件名称设置
global name         #软件名称
global title        #软件标题
```

```

global developer      #软件开发者
global ver            #软件版本号
#软件数据目录
global datapath       #数据目录
global prgpath        #软件目录
#软件平台设置信息
global winW           #默认主窗口宽度
global winH           #默认主窗口高度
global root           #窗口根句柄
global ico            #软件图标
#用户信息
global user           #用户名
global login          #用户登录标记
#绘图设置
global cns            #颜色表
global hcolor         #颜色表
global hcursor        #鼠标模式集
global ubg            #背景色
global ufg            #前景色
global utg            #文字颜色
global uvg            #成交量颜色
#小白框架控件名称    #后面更多程序代码略去，请参见源代码
    
```

9.3 全局变量初始化HP_set

通过学习，我们明白只有带模块别名的命名空间的全局变量才能对整个软件的运行期间都有效，而仅仅用 Global 定义的变量只在定义的模块文件中有效，因此模块运行结束后初始化的数据无法保存。

由于对全局变量进行初始化必须使用命名空间，定义为 g，因此在全局变量初始化模块 HP_set.py 中，每个模块都需要增加这条命令。

```
import HP_global as g
```

HP_set.py 的内容如下，由于字典定义和列表定义较长，因此这里在不影响运行的情况下省略了部分代码，完整代码见源代码。

```

import platform
import HP_global as g
import time
##数据主目录
g.datapath='\\xbdata'
g.prgpath='\\xbjq'

#软件名称
g.root=None
g.name='小白量化投资平台'
g.name='小白量化投资平台'
g.title='小白量化投资平台(1.00 版) 设计:荷蒲 QQ:2775205'
g.ico='tt.ico'
g.winW=1200
g.winH=800
g.ver=1.01
g.user='18578755056'
g.login=False
g.os=1

#白底色
g.ubg='w'
g.ufg='b'
g.utg='b'
g.uvg='#1E90FF'
#后面更多程序代码略去,请参见源代码

```

如果用户要使用量化模块独立运行,不依赖于小白量化框架,就必须先加下面两行命令。

```

import HP_global as g
from HP_set import *

```



注意

在小白量化框架中,系统先自动运行“from HP_set import *”命令,即已经对系统 g 命名空间的全局变量作了初始化。当用户回测代码时,不需要再使用这



条命令，因为在程序运行过程中很多变量参数已经发生改变，如股票代码、当前画面、选择的指标、用户输入的回测起始日期和终止日期等。如果再使用这条命令，所有变量就会重置。

在后面模块中，我们都用符号“#”注释了这条命令。如果需要独立运行模块，就取消注释符。

9.4 本地数据及格式处理HP_data

在使用股票数据时，我们会发现“index,date,close,open,low,high,volume”这种 DataFrame 数据格式最容易进行数据处理。

一般数据文件都非常大而程序容量很小，所以经常需要备份。但由于数据目录不需要频繁备份，因此可以把数据存储磁盘目录与运行程序目录进行分离。

数据目录为独立子目录 xldata，如 d:\xldata。

本地数据的读取或保存都是默认为\xldata 目录，需要与量化软件在同一个硬盘分区中。

有关数据处理的算法文件都放到 HP_data.py 模块中。

小白量化投资平台的股票数据格式如下：

index（默认整型数据索引），列名（不分顺序）为 date（或 datetime），close，open，low，high，volume。

这个格式与 Tushare 股票数据格式接近，因此，无论读取什么数据格式都必须先作转换，即转换为小白股票数据格式。

小白量化数据的目录格式：

主目录为\xldata。

下面的子目录如下：

cp：彩票数据。

day：日线数据。

minute：分钟线数据。

month：月线数据。

week：周线数据。

temp: 系统临时数据。

数据文件有股票基本数据、实时数据、财务数据等。

数据获取方式仿照 Tushare 获取数据的函数。如果仅用本地数据, 就可以替换 Tushare 模块。例如:

```
import HP_global as g
from HP_set import *
#import tushare as ts
import HP_data as ts

df1=ts.get_growth_data(2002,1)
print(df1)
df2 = ts.get_k_data('600080',ktype='D')
print(df2)
```

为了不产生冲突, 我们还是建议这样使用。

```
import HP_global as g
from HP_set import *
import HP_data as hp
df2 = hp.get_k_data('600080',ktype='D')
print(df2)
```

由于 Tushare 股票数据只有 6 位代码, 没有后缀, 因此需要增加 index 来表示是否为指数代码。

Tushare 获取 K 线数据:

```
ts.get_k_data(code=None, start='', end='', ktype='D', autype='qfq',
index=False, retry_count=3, pause=0.001)
```

下载 ts 股票历史数据.py 的代码, 见示例 9-1。

```
#下载全部数据, 包含日线、周线、分时线, 大约需要 7 小时
import tushare as ts
import HP_global as g
from HP_set import *
import time
st=ts.get_stock_basics() #获取全部股票代码
print(len(st))
```

```

index=False                                #不是指数
autype='None'
ktype='D'
pp=''
if autype=='None':
    pp=pp+'none\\'
if autype=='hfq':
    pp=pp+'hfq\\'
if index==True:
    pp=pp+'index'
kk=''
if ktype=='D':
    kk=kk+'\\day\\'
if ktype=='W':
    kk=kk+'\\week\\'
if ktype=='M':
    kk=kk+'\\month\\'
if ktype=='5':
    kk=kk+'\\minute\\5\\'
ds='1991-01-01'
de=time.strftime('%Y-%m-%d',time.localtime(time.time()))
print(ds,de)
i=0
for ss in st.index:
    i=i+1
    print(i,ss)
    for autype in ['qfq','None','hfq']:
        for ktype in ['D','W','M']:
            pp=''
            if autype=='None':
                pp=pp+'none\\'
            if autype=='hfq':
                pp=pp+'hfq\\'
            if index==True:
                pp=pp+'index'
            kk=''
            if ktype=='D':

```

```

        kk=kk+'\\day\\'
    if ktype=='W':
        kk=kk+'\\week\\'
    if ktype=='M':
        kk=kk+'\\month\\'
    if ktype=='5':
        kk=kk+'\\minute\\5\\'
    df1 = ts.get_k_data(ss,ktype=ktype,start=ds,end=de,index=
        d index,autype=autype)
    ssl=kk+pp+ss+'.csv'
    print(ssl)
    df1.to_csv(g.datapath+ssl, encoding = 'gbk')

```

下面给出 HP_data.py 的部分代码，其他请参见源代码。

```

from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import datetime as dt
from matplotlib import dates as mdates
import HP_global as g
from HP_set import *
##数据处理
#g.datapath = '\\xxdata'
#数字股票代码转换字符串股票代码
def stockname(n):
    '''
    函数说明
    数字股票代码转换字符串股票代码
stockname(n)
    参数: n 整型
    返回: 字符串
    '''
    s=str(n) #把数字转为字符串
    s=s.strip() #删除字符串前后空格
    if (len(s)<6 and len(s)>0): #如果字符串长度为1~5,前面用0补够6位长度
        s=s.zfill(6)+'.SZ' #深圳股后缀加.SZ
    if len(s)==6: #上海股票一般为100000以上的数字。

```



```

        if s[0:1]=='0':
            s=s+'.SZ'
        else:
            s=s+'.SH'

    return s

def jqtotots(df1):
    a=[x.strftime("%Y-%m-%d") for x in df1.index]
    df1.insert(0,'date',a)
    df1=df1.reset_index(level=None, drop=True ,col_level=0, col_fill='')
    return df1

def tstojq(df1):
    a=[dt.datetime.strptime(x,'%Y-%m-%d') for x in df1['date']]
    df1.insert(0,'date2',a)
    df1=df1.reset_index(level=None, drop=True ,col_level=0, col_fill='')
    df1.index=df1['date2']
    del df1['date2']
    del df1['date']
    return df1
#其他代码参见源代码
    
```

还有很多不同的数据格式转换，读者可以在源代码中下载。

9.5 公式基础函数库HP_formula

如果要想实现更多大智慧和通达信指标的移植，就需要有大量的公式基础函数库支持，这些函数都放到 HP_formula.py 模块文件中，这个模块可逐步增加新函数来进行完善。

公式基础函数主要是 Series 类函数，放在 HP_formula.py 模块文件的前半部分；后半部分是一些常用股票公式的算法。由于文件内容较多，这里只给出核心代码，其他请参照源代码。

```

#仿通达信大智慧公式基础库 Ver1.00
#版本: Ver1.00
#设计人: 独狼荷蒲
#百度: 荷蒲指标
#开始设计日期: 2018-9-17
    
```



```
#主程序: HP_main.py
"""*****
通达信公式转为 Python 公式的过程:
1. “:=” 为赋值语句, 用程序把 “:=” 替换为 Python 的赋值命令 “=”。
2. “:” 为公式的赋值带输出画线命令, 把 “:” 替换为 ‘=’, ‘:’ 前为输出变量, 顺序写到
return 返回参数中。
3. 全部命令转为英文大写。
4. 删除绘图格式命令。
5. 删除每行末的分号。
6. 参数可写到函数参数表中. 例如, def KDJ(N=9, M1=3, M2=3):

例如, 通达信 KDJ 指标公式描述如下。
参数表 N:=9, M1:=3, M2:=3
RSV:=(CLOSE-LLV(LOW,N))/(HHV(HIGH,N)-LLV(LOW,N))*100;
K:SMA(RSV,M1,1);
D:SMA(K,M2,1);
J:3*K-2*D;
#改为 Python 代码如下:
def KDJ(N=9, M1=3, M2=3):
    RSV = (CLOSE - LLV(LOW, N)) / (HHV(HIGH, N) - LLV(LOW, N)) * 100
    K = EMA(RSV, (M1 * 2 - 1))
    D = EMA(K, (M2 * 2 - 1))
    J = 3*K-2*D
    return K, D, J
#####基本函数库#####
"""
import pandas as pd
import numpy as np
import HP_global as g

def EMA(Series, N):
    return pd.Series.ewm(Series, span=N, min_periods=N - 1, adjust=
True).mean()

def MA(Series, N):
    return pd.Series.rolling(Series, N).mean()
#后面更多程序代码略去, 请参见源代码
```

在引入模块文件时一般都设置别名，但是读者在使用 HP_formula.py 模块文件时，不要用别名。

导入股票函数模块的方式：

```
from HP_formula import *
```

当用户写自编公式时可以省略别名。

下面改编一个大智慧荷蒲能量指标，见示例 9-2。

```
import HP_global as g
from HP_set import *
import HP_data as hp
from HP_formula import *

#首先要对数据预处理
df = hp.get_k_data('600080',ktype='D')
mydf=df.copy()
CLOSE=mydf['close']
LOW=mydf['low']
HIGH=mydf['high']
OPEN=mydf['open']
VOL=mydf['volume']
C=mydf['close']
L=mydf['low']
H=mydf['high']
O=mydf['open']
V=mydf['volume']

#荷蒲能量指标，判断股价上涨的能力
'''
#荷蒲能量指标，大智慧指标
VAR1:=(HIGH+LOW+CLOSE)/3;
CZ1:=SUM(MAX(0,HIGH-REF(VAR1,1)),26)/SUM(MAX(0,REF(VAR1,1)-LOW),26)*100;
CZ:=EMA(CZ1,4);
AA1:=SUM(IF(CLOSE>REF(CLOSE,1),VOL,IF(CLOSE<REF(CLOSE,1),-VOL,0)),0);
MMNL:CZ*AA1 ,COLORRED;
ma5:ma(mmn1,5);
'''
#下面改为 Python 函数
def hpn1():
```

```

VAR1=(HIGH+LOW+CLOSE)/3          #计算股价重心
mydf['Z0']=0                      #因为一些函数无法处理纯数字，所以必须把数字改为pd.Series
A1=MAX(mydf['Z0'],HIGH-REF(VAR1,1))
CZ1=SUMX(A1,26)
CZ2=SUMX(MAX(mydf['Z0'],REF(VAR1,1)-LOW),26)
CZ3=CZ1/CZ2
CZ=EMA(CZ1,4)
B1=IF(CLOSE<REF(CLOSE,1),-VOL,mydf['Z0'])
B2=IF(CLOSE>REF(CLOSE,1),VOL,B1)
AA1=SUM(B2,len(B2)) #在股票软件公式中的SUM(),MA(),COUNT()等方法中，参数0表示全部
HPNL=CZ*AA1
MA5=MA(HPNL,5)
return HPNL,MA5
x1,x2=hpn1()                      #指标用法，其中 x1>x2 表示有上涨能力
mydf = mydf.join(pd.Series(x1,name='X1')) #增加 x1 到 mydf 中
mydf = mydf.join(pd.Series(x2,name='X2')) #增加 x2 到 mydf 中
mydf=mydf.tail(200)               #显示最后 200 条数据线
#下面是绘线语句
mydf.X1.plot.line(legend=True)
mydf.X2.plot.line(legend=True)

```

软件运行结果见图 9-1。

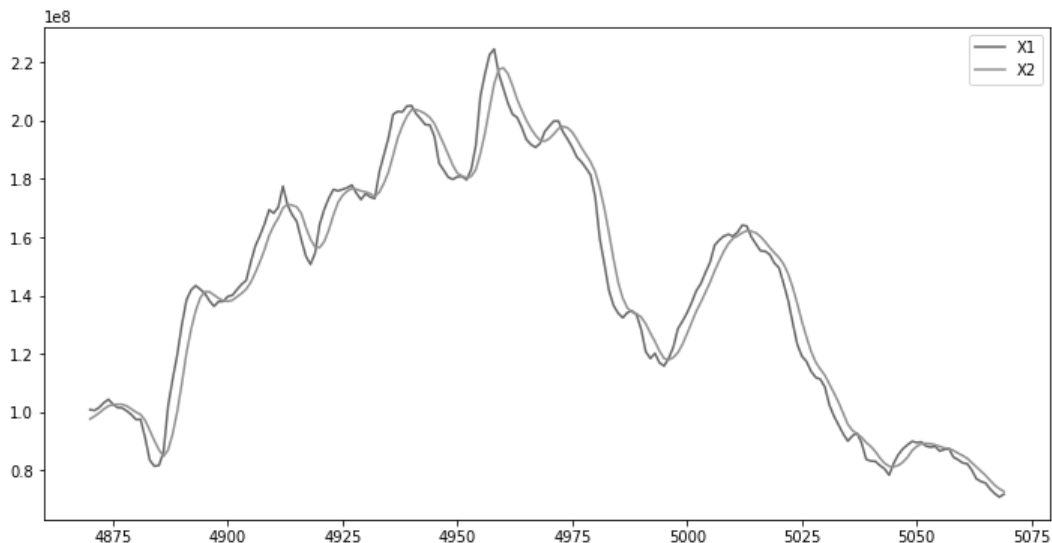


图 9-1 示例程序运行结果

9.6 窗口容器库HP_view

在窗口容器库 HP_view.py 中存放的是关于容器布局的类或函数。

所有的类和函数都有上级窗口或者容器的变量名，变量名存储的是窗口对象或容器对象的返回值。在 HP_view.py 中的类或函数接收这个变量名并做布局设置，因此，读者需要建立新的窗口或容器布局，并放到 HP_view.py 模块文件中。

在 HP_view.py 模块文件的主要代码中，布局总类很多，这里仅展示典型的布局类。例如，窗口分割布局有 2 窗口横排、2 窗口竖排、3 窗口、4 窗口等。指标线绘图窗口仅显示绘制单 K 线就有 4 种方式，所以整个代码较长，现只讲解和演示经典代码。HP_view.py 模块文件的主要代码如下：

```
import pandas as pd
import numpy as np
from matplotlib import dates as mdates
from matplotlib import ticker as mticker
#from matplotlib.finance import candlestick_ohlc #matplotlib2.0.0 用
from mpl_finance import candlestick_ohlc #matplotlib3.0.0 用
#from matplotlib.dates import DateFormatter, WeekdayLocator, DayLocator,
MONDAY, YEARLY
#from matplotlib.dates import MonthLocator, MONTHLY
import matplotlib
import matplotlib.pyplot as plt
#from matplotlib.backends.backend_tkagg import
FigureCanvasTk, NavigationToolbar2Tk #matplotlib 2.0.2
from matplotlib.backends.backend_tkagg import (
FigureCanvasTkAgg, NavigationToolbar2Tk) ##matplotlib 3.0.2
#matplotlib.use('TkAgg') #只有 matplotlib 2.0.2 需要，高版本可不用设置
from matplotlib.figure import Figure
import datetime as dt
import tkinter as tk
from tkinter import ttk
from tkinter import *
```

```

from PIL import Image, ImageTk
import HP_global as g
import HP_lib as mylib
import HP_draw as mydraw
import HP_data as hp
from HP_draw import *
plt.rcParams['font.sans-serif']=['SimHei']           #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False           #用来正常显示负号
#####
# 1 #
#####
# 2 #
#####
class view2(Frame):                                #继承 Frame 类
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.root = master                        #定义内部变量 root
        self.m=2
        self.v=[]
        self.v1=tk.Frame(self)
        self.v2=tk.Frame(self)
        self.v1.pack(side=tk.TOP, fill=tk.BOTH, expand=1,ipady=1,pady=1,
ipadx=1,padx=1)
        self.v2.pack(side=tk.BOTTOM, fill=tk.BOTH,
expand=1,ipady=1,pady=1, ipadx=1,padx=1)
        self.v.append(self.v1)
        self.v.append(self.v2)

#.....

def axview1(v,df,t,n=2):
    #显示 K 线，带 6 条均线
    df2=df.copy()
    df2.dropna(inplace=True)                      #删除无效数据
    date_tickers=df2['date']                      #刻度值
    del df2['date']
    df2['date']=df2.index

```

```

ma5=pd.Series.rolling(df2.close, 5).mean()      #股票收盘价 5 日均线
ma10=pd.Series.rolling(df2.close, 10).mean()     #股票收盘价 10 日均线
fig = plt.figure(facecolor=g.ubg,figsize=(1,1))
ax=plt.subplot(fc=g.ubg)
days = df2.reindex(columns=['date','open','high','low','close'])
ax.set_xticks(range(len(date_tickers)))
ax.set_xticklabels(date_tickers)
ax.tick_params(axis='y', colors=g.ufg)
ax.tick_params(axis='x', colors=g.ufg)
plt.ylabel('Stock price')
candlestick_ohlc(ax, days.values, width=.6, colorup='#ff1717',
colordown='#53c156')
ax.plot(days.date.values,ma5,label='MA5', linewidth=1.5)
ax.plot(days.date.values,ma10,label='MA10', linewidth=1.5)
if n>=3:
    ma20=pd.Series.rolling(df2.close, 20).mean() #股票收盘价 20 日均线
    ax.plot(days.date.values,ma20,label='MA20', linewidth=1.5)
if n>=4:
    ma30=pd.Series.rolling(df2.close, 30).mean() #股票收盘价 30 日均线
    ax.plot(days.date.values,ma30,label='MA30', linewidth=1.5)
if n>=5:
    ma60=pd.Series.rolling(df2.close, 60).mean() #股票收盘价 60 日均线
    ax.plot(days.date.values,ma60,label='MA60', linewidth=1.5)
if n>=6:
    ma120=pd.Series.rolling(df2.close, 120).mean() #股票收盘价 120 日均线
    ax.plot(days.date.values,ma120,label='MA120', linewidth=1.5)
ax.grid(True, color='r')
ax.xaxis.set_major_locator(mticker.MaxNLocator(8)) #x 轴分成几等分
ax.xaxis.label.set_color(g.ufg)
ax.yaxis.label.set_color(g.ufg)
plt.legend()      #显示图中右上角的提示信息
plt.suptitle(t,color=g.ufg)
plt.subplots_adjust(left=.06, bottom=.08, right=.96, top=.96,
wspace=.15, hspace=0.1)
plt.close()      #关闭窗口
canvas =FigureCanvasTkAgg(fig, master=v)      #设置 tkinter 绘图区
canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=1)

```

```
return ax
```

#后面更多程序代码略去，请参见源代码

使用容器类，首先把窗口划分为4个子容器画面，然后将第4个（下标为3）子容器再划分为3个子容器画面，将划分的第3个子容器再划分为4个子容器画面。为了让大家能够看清楚划分的结果，我们对所有子容器填充不同颜色，此例子也表明 Tkinter 的容器类可以重复嵌套。在嵌套过程中，要保存子容器的 id，因为在总布局完成后，需要利用这些 id 来输出想要的画面或信息。

为了便于管理容器类“view”，我们增加了类属性“m”，用于存放子画面的总数；增加了列表属性“v”，用于存放子画面的 id。下面见容器类示例 9-3。

```
import tkinter as tk
import HP_global as g
import HP_set
from HP_view import *
import HP_data as hp

root = tk.Tk()
root.title('我的新建窗口')
setCenter(root,1200,800)
#把字典颜色转化为列表
colors=[]
for (d,x) in g.cns.items():
    colors.append(x)
print(len(colors))
#把窗口划分为4个子容器
myview=view4(root)
print('窗口划分子窗口数：',myview.m)
#给所有子容器填充不同颜色
for i in range(0,myview.m-1):
    myview.v[i].config(bg=colors[i])
    tk.Label(myview.v[i],text='myview.v[%d]'%i,width=12).pack()
myview.pack(fill=tk.BOTH, expand=1)
#将第4个子容器再划分为3个子容器
myview2=view3(myview.v[3])
#给所有子容器填充不同颜色
```

```
for i in range(0,myview2.m-1):
    myview2.v[i].config(bg =colors[i+4])
    tk.Label(myview2.v[i],text='myview2.v[%d]'%i,width=12).pack()
myview2.pack(fill=tk.BOTH, expand=1)
#将所划分的第 3 个子容器再划分为 4 个子容器
myview3=view4(myview2.v[2])
#给所有子容器填充不同颜色
for i in range(0,myview3.m):
    myview3.v[i].config(bg =colors[i+12])
    tk.Label(myview3.v[i],text='myview3.v[%d]'%i).pack()
myview3.pack(fill=tk.BOTH, expand=1)
root.mainloop()
```

示例运行结果见图 9-2。

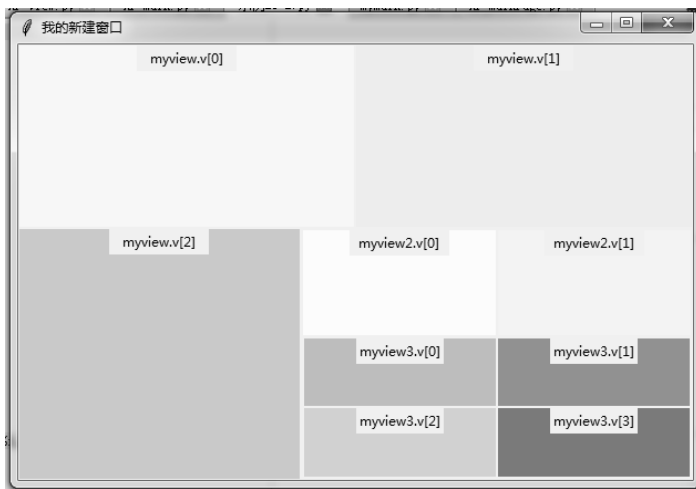


图 9-2 示例程序运行结果

9.7 指标绘图库HP_draw

本节介绍指标绘图库 HP_draw.py 模块文件，并对之前所学知识进行汇总，让读者可以做出类似流行股票软件所呈现的图形画面。

一些绘图类的参数已放到全局变量中。


```
#绘图设置
global ubg #背景色
global ufg #前景色
global utg #文字颜色
global uvg #成交量颜色
```

数据定义如下:

```
#白底色
#g.ubg='w'
#g.ufg='b'
#g.utg='b'
#g.uvg='#1E90FF'

#黑底色
g.ubg='#07000d'
g.ufg='w'
g.utg='w'
g.uvg='#FFD700'
```

底色主要是白底色和黑底色,读者可以根据需要增加新的底色。

Python 中的颜色可以使用英文表示,如 red, blue, yellow 等,也可以使用 RGB 颜色模式“#RRGGBB”表示,其中“R”表示红色,二进制“00-FF”,“G”是绿色,“B”是蓝色。

二进制值“00-FF”对应十进制值“0-255”,我们可以用如下方式来转换。

```
r=10
g=255
b=0
color='#%02x%02x%02x'%(r,g,b)
```

color 变量的内容是字符串“#64ff00”,也就是 Python 中的颜色表示。

我们定义的上百种颜色都存放在字典“g.cns”中,读者可以通过字典“key”来获取对应的颜色值。例如,由 g.cns['gold']可以得到#FFD700 值。

在 Matplotlib 模块绘图时,先要建立绘图画板 fig,并在这个画板上划分出子画板 ax, ax 可理解为带坐标系的子画板。

如果 fig 只有一个画板,就要用 ax=plt.subplot(000)。

多子图可以使用 `ax1=plt.subplot(210)`, `ax2=plt.subplot2grid((7,4)或(0,0), rowspan=5, colspan=4)` 方式来划分 `ax` 子图的显示方式。

我们也可以将子图划分存放在窗口容器库 `HP_view.py` 模块中, 模块中的函数如下:

```
#显示两个指标图, 其中K线算一个指标
def axview2x(v, df, t, n=2, f='VOL'):
    #叠加成交密度的K线
    ...
    return ax

#其中, 参数v是tkinter的子窗口名字id, df是股票数据库
#t是ax画面标题, n代表K线上收盘价均线数量, n最大为6
#f是指标名称, 如VOL, KDJ, MACD等
#返回ax的id
```

凡是入口使用 Tkinter 窗口或容器名“id”的绘图类函数, 都在窗口容器库 `HP_view.py` 模块文件中。

凡是入口使用 `matplotlib.pyplot` 的 `ax` 子画板的绘图类函数, 都在指标绘图库 `HP_draw.py` 模块文件中。

窗口布局使用窗口容器库 `HP_view.py` 模块, 并可放置增加的新布局文件。

指标绘图函数都在指标绘图库 `HP_draw.py` 模块文件中, 并可放置新增加的指标绘图函数。

窗口、容器和子图的包含关系见图 9-3。

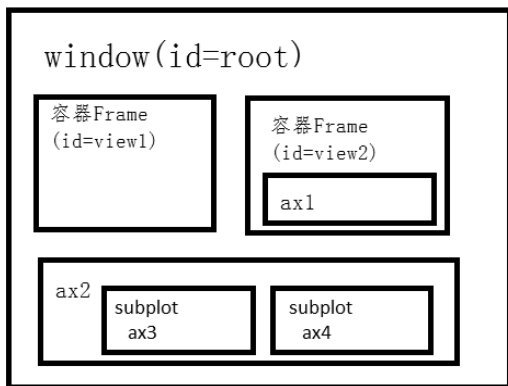


图 9-3 窗口、容器和子图的包含关系

下面是指标绘图库 HP_draw.py 模块文件的部分代码。

```

#-*- coding: utf-8 -*-
from pandas import DataFrame, Series
import pandas as pd; import numpy as np
import matplotlib.pyplot as plt
from matplotlib import dates as mdates
from matplotlib import ticker as mticker
#from matplotlib.finance import candlestick_ohlc
from mpl_finance import candlestick_ohlc
from matplotlib.dates import DateFormatter, WeekdayLocator, DayLocator,
MONDAY, YEARLY
    from matplotlib.dates import MonthLocator,MONTHLY
import matplotlib
from numpy import arange, sin, pi
#from matplotlib.backends.backend_tkagg import
FigureCanvasTk,NavigationToolbar2Tk #matplotlib 2.0.2
    from matplotlib.backends.backend_tkagg import (
FigureCanvasTkAgg, NavigationToolbar2Tk) ##matplotlib 3.0.2
from matplotlib.backend_bases import key_press_handler
from matplotlib.figure import Figure
import datetime as dt
import pylab
import matplotlib
from PIL import Image, ImageTk
import HP_zwdata as sd
import HP_global as g
import HP_lib as mylib
from HP_formula import *

#在 ax 区绘制 K 线
def ax_K(ax,df,t,n=2):
    #显示 K 线,带 6 条均线
    df2=df.copy()
    df2.dropna(inplace=True) #删除无效数据
    date_tickers=df2['date'] #刻度值
    del df2['date']
    df2['date']=df2.index

```

```

ma5=pd.Series.rolling(df2.close, 5).mean() #股票收盘价 5 日均线
ma10=pd.Series.rolling(df2.close, 10).mean() #股票收盘价 10 日均线
#fig = plt.figure(facecolor=g.ubg,figsize=(1,1))
#ax=plt.subplot(fc=g.ubg)
#fig, ax = plt.subplots()
days = df2.reindex(columns=['date','open','high','low','close'])
ax.set_xticks(range(len(date_tickers)))
ax.set_xticklabels(date_tickers)
ax.tick_params(axis='y', colors=g.ufg)
ax.tick_params(axis='x', colors=g.ufg)
plt.ylabel('Stock price')
candlestick_ohlc(ax, days.values, width=.6, colorup='#ff1717',
colordown='#53c156')
ax.plot(days.date.values,ma5,label='MA5', linewidth=1.5)
ax.plot(days.date.values,ma10,label='MA10', linewidth=1.5)
if n>=3:
    ma20=pd.Series.rolling(df2.close, 20).mean() #股票收盘价 20 日均线
    ax.plot(days.date.values,ma20,label='MA20', linewidth=1.5)
if n>=4:
    ma30=pd.Series.rolling(df2.close, 30).mean() #股票收盘价 30 日均线
    ax.plot(days.date.values,ma30,label='MA30', linewidth=1.5)
if n>=5:
    ma60=pd.Series.rolling(df2.close, 60).mean() #股票收盘价 60 日均线
    ax.plot(days.date.values,ma60,label='MA60', linewidth=1.5)
if n>=6:
    ma120=pd.Series.rolling(df2.close, 120).mean() #股票收盘价 120 日均线
    ax.plot(days.date.values,ma120,label='MA120', linewidth=1.5)
ax.grid(True, color='r')
ax.xaxis.set_major_locator(mticker.MaxNLocator(8)) #x 轴分成几等分
ax.xaxis.label.set_color(g.ufg)
ax.yaxis.label.set_color(g.ufg)
plt.legend() #显示图中右上角的提示信息
plt.suptitle(t,color=g.ufg)
#plt.subplots_adjust(left=.06, bottom=.08, right=.96, top=.96,
wspace=.15, hspace=0.1)
plt.show()
return ax

```

```

#在 ax 区绘制 VOL
def ax_VOL(ax1,df):
    df2=df.copy()
    v1=pd.Series.rolling(df2.volume, 5).mean() #5 日均线
    v2=pd.Series.rolling(df2.volume, 10).mean() #10 日均线
    rsiCol = '#c1f9f7'
    posCol = '#386d13'
    ax1.plot(df2.date.values, v1, rsiCol, linewidth=1,label="VMA5")
    ax1.plot(df2.date.values, v2, posCol, linewidth=1,label="VMA10")
    ax1.bar(df2.date.values,df2.volume.values, facecolor=g.uvg, alpha=.8)
    ax1.yaxis.label.set_color(g.ufg)
    plt.gca().yaxis.set_major_locator(mticker.MaxNLocator(prune='upper'))
    ax1.tick_params(axis='y', colors=g.ufg)
    ax1.tick_params(axis='x', colors=g.ufg)
    ax1.yaxis.set_major_locator(mticker.MaxNLocator(nbins=4, prune='upper'))
    ax1.tick_params(axis='x', colors=g.ufg)
    plt.ylabel('volume')
    plt.legend() #显示图中右上角的提示信息
    ax1.grid(True, color='r')
    ax1.yaxis.set_major_locator(mticker.MaxNLocator(nbins=6, prune='upper'))
    return ax1

#在 ax 区绘制 KDJ
def ax_KDJ(ax1,mydf):
    CLOSE=mydf['close']
    LOW=mydf['low']
    HIGH=mydf['high']
    OPEN=mydf['open']
    VOL=mydf['volume']
    C=mydf['close']
    L=mydf['low']
    H=mydf['high']
    O=mydf['open']
    V=mydf['volume']

```

```
#df 格式必须使用 tushare 数据格式
df=mydf.copy()
#KDJ Python 随机指标
def KDJ(N=9, M1=3, M2=3):
    RSV = (CLOSE - LLV(LOW, N)) / (HHV(HIGH, N) - LLV(LOW, N)) * 100
    K = SMA(RSV,M1,1)
    D = SMA(K,M2,1)
    J = 3*K-2*D
    return K, D, J

#使用 KDJ 指标, 返回 K, D, J 序列
K,D,J=KDJ(9,3,3)

df = df.join(pd.Series( K,name='K'))
df = df.join(pd.Series( D,name='D'))
df = df.join(pd.Series( J,name='J'))

ax1.plot(df.date.values, df.K.values, color= 'green', lw=2,label="K")
ax1.plot(df.date.values,df.D.values, color= 'red', lw=2,label="D")
ax1.plot(df.date.values, df.J.values, color= 'blue', lw=2,label="J")
plt.gca().yaxis.set_major_locator(mticker.MaxNLocator(prune='upper'))
ax1.tick_params(axis='x', colors=g.utg)
ax1.tick_params(axis='y', colors=g.utg)
ax1.grid(True, color='r')
plt.ylabel('KDJ', color=g.utg)
plt.legend() #显示图中右上角的提示信息
ax1.yaxis.set_major_locator(mticker.MaxNLocator(nbins=6, prune='upper'))
return ax1

def ax_RSI(ax2,days):
    x=6
    y=12
    z=24
    rsiCol = '#c1f9f7'
    posCol = '#386d13'
    negCol = '#8f2020'
    df=mylib.RSIX(days,x,'RSI1')
```

```

df=mylib.RSIX(df,y,'RSI2')
df=mylib.RSIX(df,z,'RSI3')
fillcolor = '#00ffe8'
ax2.plot(df.date.values, df.RSI1.values, color=rsiCol, lw=2,
label="$RSI1$")
ax2.plot(df.date.values, df.RSI2.values, color=posCol, lw=2,
label="$RSI2$")
ax2.plot(df.date.values, df.RSI3.values, color=negCol, lw=2,
label="$RSI3$")
plt.gca().yaxis.set_major_locator(mticker.MaxNLocator(prune='upper'))
ax2.spines['bottom'].set_color("#5998ff")
ax2.spines['top'].set_color("#5998ff")
ax2.spines['left'].set_color("#5998ff")
ax2.spines['right'].set_color("#5998ff")
ax2.tick_params(axis='x', colors=g.utg)
ax2.tick_params(axis='y', colors=g.utg)
ax2.axhline(80, color=negCol)
ax2.axhline(20, color=posCol)
plt.ylabel('RSI', color=g.utg)
ax2.yaxis.set_major_locator(mticker.MaxNLocator(nbins=6, prune='upper'))
plt.legend() #显示图中右上角的提示信息
return ax2

def ax_BOLL(ax1,days):
    x=26
    rsiCol = '#c1f9f7'
    posCol = '#386d13'
    negCol = '#8f2020'
    df=mylib.BOLLX(days,x)
    fillcolor = '#00ffe8'
    ax1.plot(df.date.values, df.BOLL_u.values, color=rsiCol, lw=2,
label="$U$")
    ax1.plot(df.date.values, df.BOLL_m.values, color=posCol, lw=2,
label="$M$")
    ax1.plot(df.date.values, df.BOLL_d.values, color=rsiCol, lw=2,
label="$D$")
    ax1.plot(df.date.values, df.close.values, color='yellow', lw=3,

```

```
label="$CLOSE$")
    plt.gca().yaxis.set_major_locator(mticker.MaxNLocator
(prune='upper'))
    ax1.spines['bottom'].set_color("#5998ff")
    ax1.spines['top'].set_color("#5998ff")
    ax1.spines['left'].set_color("#5998ff")
    ax1.spines['right'].set_color("#5998ff")
    ax1.tick_params(axis='x', colors=g.utg)
    ax1.tick_params(axis='y', colors=g.utg)
    ax1.grid(True, color='r')
    plt.ylabel('BOLL', color=g.utg)
    ax1.yaxis.set_major_locator(mticker.MaxNLocator(nbins=6, prune=
'upper'))
    plt.legend() #显示图中右上角的提示信息
    return ax1

def ax_MACD(ax1,days):
    x=12
    y=26
    z=9
    rsiCol = '#c1f9f7'
    posCol = '#386d13'
    negCol = '#8f2020'
    df=mylib.MACD(days,x,y)
    fillcolor = '#00ffe8'
    ax1.plot(df.date.values, df.MACDsign_12_26.values, color=rsiCol,
lw=1,label="$DIF$")
    ax1.plot(df.date.values, df.MACD_12_26.values, color=negCol,
lw=1,label="$MACD$")
    ax1.fill_between(df.date.values, df.MACDdiff_12_26.values, 0,
alpha=0.5, facecolor=fillcolor, edgecolor=fillcolor)
    plt.gca().yaxis.set_major_locator(mticker.MaxNLocator(prune='upper'))
    ax1.spines['bottom'].set_color("#5998ff")
    ax1.spines['top'].set_color("#5998ff")
    ax1.spines['left'].set_color("#5998ff")
    ax1.spines['right'].set_color("#5998ff")
    ax1.tick_params(axis='x', colors=g.utg)
    ax1.tick_params(axis='y', colors=g.utg)
    ax1.axhline(0, color=negCol)
    ax1.grid(True, color='r')
```



```
plt.ylabel('MACD', color=g.utg)
ax1.yaxis.set_major_locator(mticker.MaxNLocator(nbins=3, prune= 'upper'))
plt.legend() #显示图中右上角的提示信息
return ax1
```

#后面更多程序代码略去, 请参见源代码

最后给出指标图形输出的演示, 见示例 9-4。

```
import time
import tkinter as tk
import HP_global as g
import HP_set
from HP_view import *
import HP_data as hp

##白底色
#g.ubg='w'
#g.ufg='b'
#g.utg='b'
#g.uvg='#1E90FF'

root = tk.Tk()
root.title('我的新建窗口')
setCenter(root,1200,800)

ds='2018-01-01'
de=time.strftime('%Y-%m-%d',time.localtime(time.time()))

#读取股票数据
df2a=hp.get_k_data('000001',ktype='D',start=ds,end=de,index=False,autype='qfq')
df2b=hp.get_k_data('600059',ktype='D',start=ds,end=de,index=False,autype='qfq')
df2c=hp.get_k_data('600030',ktype='D',start=ds,end=de,index=False,autype='qfq')
df2d=hp.get_k_data('300008',ktype='D',start=ds,end=de,index=False,autype='qfq')
```

```
#把 window 划分为 4 个子容器, 在不同子容器中显示不同股票 K 线图
xxx=view4(root)
#六均线二指标图, K 线算一个指标
axview2(xxx.v[0],df2a,'000001 六均线 K 线演示',6)

#六均线二指标图, 现实 KDJ 指标线
axview2x(xxx.v[1],df2b,'600059 六均线 K 线演示',6,'HPYYX')
#三指标图
axview3x(xxx.v[2],df2c,'600030 三指标演示')
#五指标图
axview5x(xxx.v[3],df2d,'300008 五指标演示')
xxx.pack(fill=tk.BOTH, expand=1)
root.mainloop()
```

程序运行结果见图 9-4。

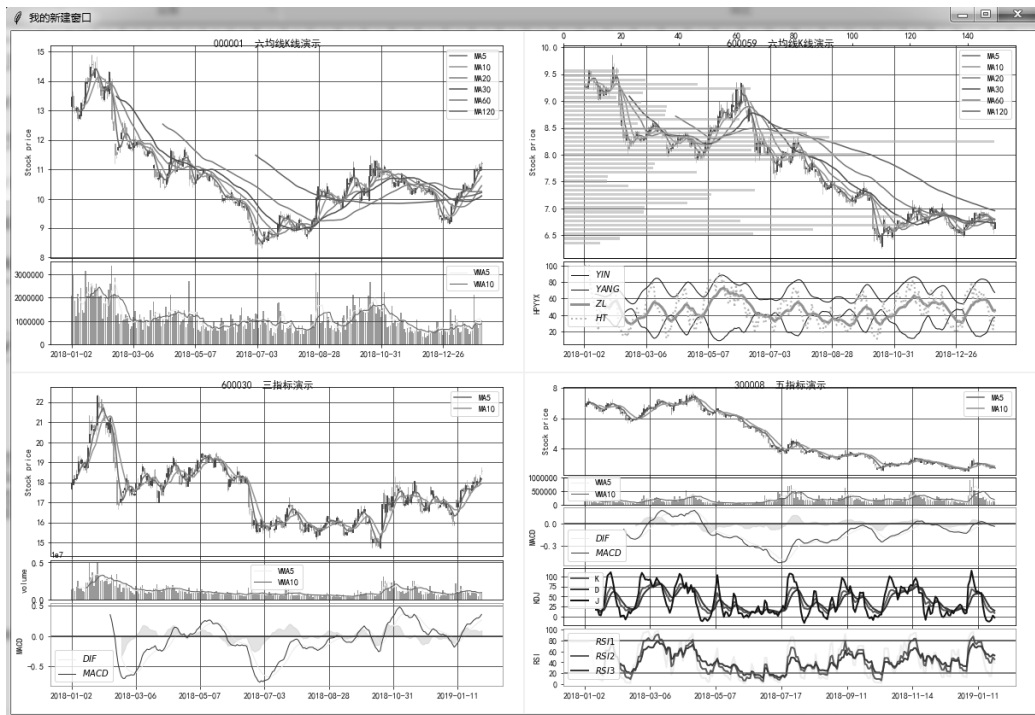


图 9-4 示例程序运行结果

9.8 回测系统库HP_sys

回测的功能都放在 HP_sys.py 模块中。

1. 股票回测原理

(1) 初始参数。输入股票代码“code”，开始日期“sdate”，结束日期“edate”，初始资金“money2”。

(2) 交易参数。印花税“stamp_duty”，即现在卖出金额的 0.001。交易佣金“trading_Commission”，一般在交易金额的 0.0002 至 0.003 之间。

(3) 交易状态。允许交易“trade”，持仓状态“position”，允许止损“stop_loss_on”，最大允许止损次数“stop_loss_max”，止损幅度“stop_loss_range”，止损价格“priceStopLoss”，证券数量“amount”，总资金“money”，最后一次买入价格“priceBuy”，最后一次卖出价格“priceSell”等。

(4) 交易过程。交易流水“trade_df”，持仓清单“security_df”。

(5) 回测过程。回测信息“text”。

根据上面的信息，我们很容易做出交易类，即把买卖算法和策略转换为买点信号“b”，卖点信号“s”。例如，当 b=1 时，如果没有持仓就发出买入命令；当 s=1 时，如果持仓就发出卖出信号。

最后根据交易原理做出 hpQuant 类。

2. 股票回测“HP_sys”模块的实现

回测系统库 HP_sys.py 模块文件的代码如下：

```
# -*- coding: utf-8 -*-
import sys,os
import numpy as np
import pandas as pd
from HP_global import *
from HP_set import *

#交易回测类
class hpQuant(object):
    def __init__(self):
```

#类初始化

```

self.order_df=None          #下单流水
self.trade_df=None         #交易流水
self.security_df=None      #持仓清单
self.money2=1000000.00     #总资金
self.money=1000000.00      #资金
self.priceBuy=0.00         #最后一次买入价格
self.priceSell=999999.00   #最后一次卖出价格
self.amount=0.00          #证券数量
self.code=""              #证券代码
self.stamp_duty=0.001      #印花税 0.1%
self.trading_Commission=0.0005 #交易佣金 0.05%
self.priceStopLoss=0.00    #止损价
self.position=False        #持仓状态
self.stop_loss_on=True     #允许止损
self.stop_loss_num=0       #当前止损次数
self.stop_loss_max=50      #最大允许止损次数，到止损次数就停止交易
self.stop_loss_range=0.05  #止损幅度
self.trade=True            #允许交易
self.Init()

def Init(self):             #初始化交易数据
    self.order_df = pd.DataFrame(columns = ['date', 'time','mode',
'code','amount','price'])
    self.order_df =self.order_df.reset_index(level=None, drop=True,
col_level=0, col_fill='')
    self.trade_df = pd.DataFrame(columns = ['date', 'time','mode',
'code','amount','price','money'])
    self.trade_df =self.trade_df.reset_index(level=None, drop=True,
col_level=0, col_fill='')
    self.security_df = pd.DataFrame(columns = ['code','amount',
'price', 'money'])
    self.security_df =self.security_df.reset_index(level=None,
drop=True, col_level=0, col_fill='')

def PrintOrder(self):       #输出下单流水
    print(self.order_df)

```

```

def PrintTrade(self):          #输出交易流水
    print(self.trade_df)

def PrintSecurity(self):      #输出持仓清单
    print(self.security_df)

def Order(self,date,time,mode,code,amount,price): #交易函数
    ln=len(self.order_df)
    df_new = pd.DataFrame({'date':date,'time':time,'mode':mode,
'code': code,'amount':amount,'price':price},index=[ln])
    self.order_df=self.order_df.append( df_new,ignore_index=True)

    ln=len(self.trade_df)
    if mode==1:                #买入
        se=amount*price*(1+self.trading_Commission)
        self.money=self.money-se
        df_new=pd.DataFrame({'date':date,'time':time,'mode':mode,'code':
code,'amount':amount,'price':price,'money':self.money},index=[ln])
        self.trade_df=self.trade_df.append(df_new,
            ignore_index=True)
        if len(self.security_df[self.security_df.code==code])==0 :
            df_new = pd.DataFrame({'code':code,'amount':amount,
                'price':se/amount, 'money':se},index=[ln])
            self.security_df=self.security_df.append(df_new,
                ignore_index=True)
        else:
            self.security_df.index=self.security_df['code']
            self.security_df.loc[code,'amount']=self.security_df.
loc[code,'amount']+amount
            self.security_df=self.security_df.reset_index(
                level=None, drop=True, col_level=0, col_fill='')

    ln=len(self.security_df[self.security_df.code==code])
    if mode==2 and ln>0:      #卖出
        self.security_df.index=self.security_df['code']
        am=self.security_df.loc[code,'amount']

        if am<amount :
            amount=am

```

```

se=amount*price*(1-self.trading_Commission-self.stamp_duty)
self.money=self.money+se
df_new = pd.DataFrame({'date':date,'time':time,
                        'mode':mode,'code': code,'amount':amount,'price':price,
                        'money':self.money},index=[ln])
self.trade_df=self.trade_df.append( df_new,
                                    ignore_index=True)
am2=self.security_df.loc[code,'amount']-amount
self.security_df.loc[code,'amount']=am2
if am2==0:
    self.security_df=self.security_df.drop(code, axis=0)
    #在行的维度上删除行
    self.security_df=self.security_df.reset_index(
        level=None, drop=True ,col_level=0, col_fill='')

#####回测功能#####
def Trade_testing(self,df,tp1,tp2,al=''):          #回测函数
    self.Init()
    self.trade=True                               #允许交易
    myMoney=self.money2
    if (al.strip()==''):
        na='property'
    else:
        na=al
    self.text='    ----开始回测-----\n'
    i = 0
    ZB_1 = []
    while i<len(df):
        close=df.close.at[i]
        if (df[tp1].at[i] >0 and self.position==False and self.trade== True):
            #买点

            self.priceBuy=close
            x=int(myMoney/(self.priceBuy*(1+self.trading_Commission))/100)
            self.amount=x*100.00
            self.Order(df.date.at[i], '14:45:01',
                        1,self.code,self.amount,close)
            myMoney=myMoney-self.amount*self.priceBuy*(1.00+self.
trading_Commission)
            self.position=True

```

```

        self.priceStopLoss=self.priceBuy*(
            1-self.stop_loss_range)
        self.text=self.text+'日期:'+df.date.at[i]
            +' 买入:'+str(round(self. amount,0))+'股,
            价格:'+str(round(self.priceBuy,2))+'\n'
        if (df[tp2].at[i] >0 and self.position==True and self.trade== True):
            #卖点

            self.priceSell=close
            myMoney=myMoney+self.amount*self.priceSell*(1.00-self.
trading_Commission-self.stamp_duty)
            self.position=False
            self.Order(df['date'].at[i], '14:45:02'
                ,2,self.code,self.amount,self.priceSell)
            self.text=self.text+'日期:'+df.date.at[i]+' 卖出:
'+str(round(self. amount,0))+'股, 价格:'+str(round(self.priceSell,2))+'获利:
'+str(round ( (myMoney-self.money2)/self.money2*100,2))+'%\n'
            self.amount=0.00

            if (close<=self.priceStopLoss and self.position==True and
self.trade and self.stop_loss_on):    #止损
                self.priceSell=self.priceStopLoss-0.01
                myMoney=myMoney+self.amount*self.priceSell*(1.00-
self.trading_Commission-self.stamp_duty)
                self.position=False
                self.stop_loss_num=self.stop_loss_num+1
                self.Order(df['date'].at[i], '14:45:02',
                    2,self.code,self.amount,self.priceSell)
                self.text=self.text+'日期:'+df.date.at[i]+' 止损:
'+str(round (self.amount,0))+'股, 价格:'+str(round(self.priceSell,2))+'获利:
'+str(round ( (myMoney-self.money2)/self.money2*100,2))+'%\n'
                self.amount=0.00
                if (self.stop_loss_num>=self.stop_loss_max):
                    self.trade=False

            y= (myMoney+self.amount*close-self.money2)/self.money2 *100
            ZB_l.append(y)
            i = i + 1

ZB_s = pd.Series(ZB_l)

```

```

        ZB = pd.Series(ZB_s, name = na)
        df = df.join(ZB)
        self.money=myMoney
        y= (myMoney+self.amount*close-self.money2)/self.money2 *100
        self.text=self.text+'总投入'+str(round(self.money2,2))+',最终获利幅度'
        '+str(round(y,0))+'\n'

    return df

```

上面为回测模块的源代码，读者可自行学习实现过程。

下面我们采用双均线策略在 5 日价格均线上穿 20 日价格均线做买点信号，在 5 日价格均线下穿 20 日价格均线做卖点信号。回测华海药业（600521）在 2017-01-01 至 2017-09-30 之间的交易收益，见示例 9-5。

```

#-*- coding: utf-8 -*-
#双均线策略回测

import pandas as pd
import numpy as np
import datetime as dt
import time
import matplotlib.pyplot as plt
import HP_global as g
import HP_lib as mylib
import HP_data as hp
from HP_sys import *
from HP_draw import *

ds='2017-01-01'                #开始日期
de='2017-09-30'                #结束日期
#de=time.strftime('%Y-%m-%d',time.localtime(time.time()))
                                #自动获取今天的日期

stockn='600521'                #股票代码,600521 华海药业
df=hp.get_k_data(stockn,ktype='D',start=ds,end=de,index=False,autype=
'qfq')

df3=df.copy()                  #深拷贝，目的是不破坏原始数据
##数据规格化
df3.dropna(inplace=True)       #删除无效值

```



```

df3=df3.reset_index(level=None, drop=True ,col_level=0, col_fill='')
#重建索引
df2=df3                                #浅拷贝，实际上是 df3 的变量指针

##双均线交易策略
df2=mylib.MA(df2,'close',5,'C5')      #把 5 日均线存放到 C5 列中
df2=mylib.MA(df2,'close',20,'C20')    #把 20 日均线存放到 C20 列中
df2=mylib.CROSS(df2,'C5','C20','B1')  #把 5 日均线上穿 20 日均线，存放列 B1，买入信号
df2=mylib.CROSS(df2,'C20','C5','S1')  #把 5 日均线下穿 20 日均线，存放列 S1，卖出信号

##回测
tt=hpQuant()                          ##初始化类
#用户可根据需要来修改下面的参数,不修改就是默认参数
#tt.money2=1000000.00                 #总资金
#tt.code=""                           #证券代码
#tt.stamp_duty=0.001                   #印花税 0.1%
#tt.trading_Commission=0.0005         #交易佣金 0.05%
#tt.stop_loss_on=True                  #允许止损
#tt.stop_loss_max=50                   #止损 3 次，就停止交易
#tt.stop_loss_range=0.05               #止损幅度

tt.code=stockn                        #证券代码，必须输入
tt.stop_loss_on=True                  #开启自动止损
df3=tt.Trade_testing(df2,'B1','S1','HL')
#开始回测，最后一个参数是返回的获利列名

print('\n 打印交易过程')
tt.PrintTrade()                      #打印交易过程
print('\n 打印持仓信息')
tt.PrintSecurity()                   #打印持仓信息
print('\n 打印内部交易记录信息')
print(tt.text)                        #打印交易信息

#绘制均线图
ax=plt.subplot(211)                   #划分为 2 个子图 ax，在 1 号子图上显示
ax_K(ax,df2,stockn,3)                #显示 K 线，设置 3 条均线
plt.suptitle(stockn)
#绘制收益图

```

```
ax2=plt.subplot(212)    #划分为2个子图ax,在2号子图上显示
df3.B1=df3.B1*2        #买入信号*2,即买点值有1变为2,为了看图清晰
df3.HL.plot(color='orange', grid=True,label="HL") #获利线
df3.B1.plot(color='red',label="B")              #买点线
df3.S1.plot(color='blue',label="S")             #卖点线
plt.legend()                                       #显示图形标记
```

程序的运行结果既有文字输出,也有图形输出。其中,文字输出如下:

打印交易过程

| | date | time | mode | ... | amount | price | money |
|----|------------|----------|------|-----|---------|----------|--------------|
| 0 | 2017-02-08 | 14:45:01 | 1 | ... | 56000.0 | 17.82300 | 1.412956e+03 |
| 1 | 2017-02-14 | 14:45:02 | 2 | ... | 56000.0 | 16.96900 | 9.502516e+05 |
| 2 | 2017-03-06 | 14:45:01 | 1 | ... | 54300.0 | 17.47800 | 7.216323e+02 |
| 3 | 2017-03-31 | 14:45:02 | 2 | ... | 54300.0 | 18.07000 | 9.804508e+05 |
| 4 | 2017-04-07 | 14:45:01 | 1 | ... | 54100.0 | 18.09400 | 1.075988e+03 |
| 5 | 2017-04-12 | 14:45:02 | 2 | ... | 54100.0 | 17.88900 | 9.674192e+05 |
| 6 | 2017-05-18 | 14:45:01 | 1 | ... | 60600.0 | 15.93400 | 1.335996e+03 |
| 7 | 2017-06-16 | 14:45:02 | 2 | ... | 60600.0 | 16.25300 | 9.847904e+05 |
| 8 | 2017-06-27 | 14:45:01 | 1 | ... | 59600.0 | 16.49300 | 1.316106e+03 |
| 9 | 2017-07-17 | 14:45:02 | 2 | ... | 59600.0 | 16.91600 | 1.007997e+06 |
| 10 | 2017-08-22 | 14:45:01 | 1 | ... | 58700.0 | 17.15600 | 4.366875e+02 |
| 11 | 2017-11-13 | 14:45:02 | 2 | ... | 58700.0 | 20.23100 | 1.186215e+06 |
| 12 | 2017-11-20 | 14:45:01 | 1 | ... | 55100.0 | 21.49900 | 1.027850e+03 |
| 13 | 2017-11-23 | 14:45:02 | 2 | ... | 55100.0 | 20.41405 | 1.124155e+06 |
| 14 | 2017-9-05 | 14:45:01 | 1 | ... | 54300.0 | 20.68700 | 2.890322e+02 |

[15 rows x 7 columns]

打印持仓信息

| | code | amount | price | money |
|---|--------|---------|-----------|--------------|
| 0 | 600521 | 54300.0 | 20.697343 | 1.123866e+06 |

打印内部交易记录信息

----开始回测----

日期:2017-02-08 买入:56000.0 股, 价格:17.82

日期:2017-02-14 卖出:56000.0 股, 价格:16.97 获利:-4.97%

日期:2017-03-06 买入:54300.0 股, 价格:17.48

日期:2017-03-31 卖出:54300.0 股, 价格:18.07 获利:-1.95%
 日期:2017-04-07 买入:54100.0 股, 价格:18.09
 日期:2017-04-12 卖出:54100.0 股, 价格:17.89 获利:-3.26%
 日期:2017-05-18 买入:60600.0 股, 价格:15.93
 日期:2017-06-16 卖出:60600.0 股, 价格:16.25 获利:-1.52%
 日期:2017-06-27 买入:59600.0 股, 价格:16.49
 日期:2017-07-17 卖出:59600.0 股, 价格:16.92 获利:0.8%
 日期:2017-08-22 买入:58700.0 股, 价格:17.16
 日期:2017-11-13 卖出:58700.0 股, 价格:20.23 获利:18.62%
 日期:2017-11-20 买入:55100.0 股, 价格:21.5
 日期:2017-11-23 止损:55100.0 股, 价格:20.41 获利:9.42%
 日期:2017-9-05 买入:54300.0 股, 价格:20.69
 总投入 1000000.0, 最终获利幅度 36.0%

图形输出见图 9-5。

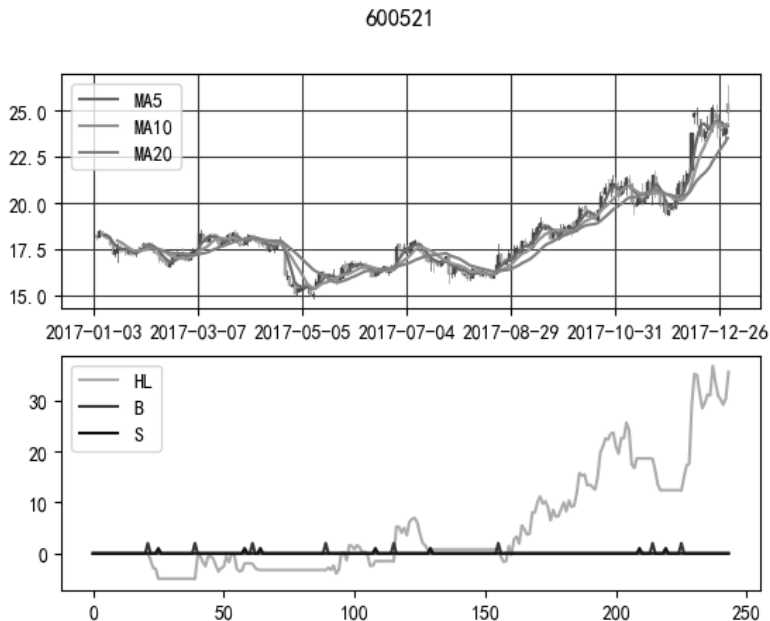


图 9-5 示例程序运行结果

通过本节的学习,读者能掌握回测系统的建立过程。这里仅仅是实现了单只股票的回测,读者可以尝试修改回测类,设计一个多支股票操作的回测程序。

9.9 智能聊天对话系统HP_robot

1. 智能聊天对话系统的功能

小白智能聊天对话系统是一个智能输入输出系统，即智能人机对话系统。下面是实现的部分功能。

(1) 基于知识库的人机帮助功能。我们把知识库以问答的形式存放在 FAQ.txt 文件中，而 FAQ.txt 文件必须是 UTF-8 的、无 bom 格式的文本文件。

注释：注释文字由符号“#”开头，且一行都是注释内容。

问答块格式如下：

【问题】问题标题

“【问题】”可以有一个或多个，至少有一个，且必须由“【问题】”开头。

答案内容可以有多行，但必须紧跟着上面的“【问题】”行，多行答案中间不能有空白的行，多个问答块之间可以用空白行分割。

(2) 中文处理使用了 jieba 库。根据提问文字，找出知识库里的“【问题】”和输入句子相似度最高的句子，然后返回对应的答案。例如，输入：股票是啥。

系统会根据“【问题】”相似度得到股票的定义。

【问题】什么是股票？

【问题】股票是什么？

【问题】股票？

股票是一种由股份有限公司签发的用以证明股东所持股份的凭证，它表明股票的持有者对股份公司的部分资本拥有所有权。由于股票包含经济利益，且可以上市流通转让，因此也是一种有价证券。

(3) 如果无法找到合适的答案，就通过互联网把问题抛给网络聊天机器人图灵。例如，输入：今天广州天气？

我:2019-04-01 00:28:32

股票是啥

通通:2019-04-01 00:28:32

股票是一种由股份有限公司签发的用以证明股东所持股份的凭证，它表明股票的持有者对股份公司的部分资本拥有所有权。由于股票包含经济利益，且可以上市流通转让，因此也是一种有价

证券。

我:2019-04-01 00:28:41

今天广州天气

通通:2019-04-01 00:28:41

广州:周一, 阵雨转多云 东北风 3-4 级, 最低气温 18 摄氏度, 最高气温 22 摄氏度

(4) 输入框可以直接与网上聊天机器人对话。

@开头文字, 是发给茉莉聊天机器人的对话。

*开头文字, 是发给图灵聊天机器人的对话。

(5) 因为程序运行中无法输出 `print()` 语句信息, 所以我们需要把程序运行中的输出信息全部输入到聊天信息窗口中。

以“g”命名的空间全局变量中, `g.ttmsg` 代表信息输出窗口, `g.mymsg` 代表信息输入窗口。

下面函数可以直接输出信息到聊天窗口。

```
#用户输出信息
def tprint(txt):
    g.ttmsg.insert(END, txt)
    g.ttmsg.see(END)

#用户输出信息,带颜色
def ttprint(txt,color):
    g.ttmsg.insert(END, txt,color)
    g.ttmsg.see(END)
```

(6) 运行 Python 代码。

默认以“>”开头的程序全部都是 Python 代码。例如, 把下面信息输入聊天对话框, 程序会让两个网络机器人互相聊天。

```
>
talk="我们开始聊天吧!"    #用户可以修改话题
for i in range(20):
    msgcontent = '通通:'+time.strftime("%Y-%m-%d %H:%M:%S",time.localtime())+'\n '
    ttprint(msgcontent, 'green')
    tprint(' '+talk+'\n')
```

```
#茉莉聊天机器人接口
talka=moli_robot(talk)
msgcontent = '茉莉:'+time.strftime("%Y-%m-%d %H:%M:%S",time.localtime())+'\n '
ttprint(msgcontent, 'red')
tprint(' '+talka+'\n')

#通通聊天机器人接口
talk=tuling_robot(talka)
#time.sleep(2)          #暂停 3 秒
```

2. 多线程运行 Python 代码

如果用户的代码中包含延时代码，就会引起整个系统卡顿。因此，我们对比较占用系统时间的代码采用多线程的方式运行。

构造多线程的函数：

```
#多线程启动函数
def thread_it(func, *args):
    '''将函数打包进线程'''
    #创建
    t = threading.Thread(target=func, args=args)
    #守护 !!!
    t.setDaemon(True)
    #启动
    t.start()

#运行用户代码
def EXEC(st):
    try:
        exec(st)
        return('命令运行完成。')
    except Exception as e:
        return('用户命令出错:'+str(e))
    return('命令运行完成。')
```

启动用户 Python 的相关代码:

```
xx2=xx2.strip()
thread_it(EXEC,xx2)
```

3. 智能聊天模块 HP_robot 代码

下面给出 HP_robot.py 的部分源代码。

```
import urllib.request
import urllib.parse
import json
import time
import gzip
import random
import hashlib
import jieba
import jieba.posseg as pseg
import requests
import threading
from tkinter import *
from tkinter.messagebox import *
from tkinter.filedialog import *
from threading import Timer
import HP_global as g

#判断是否是英文句子
def isenglish(ss):
    result=True
    for c in ss.lower():
        if c in "abcdefghijklmnopqrstuvwxyz,.' !?":
            continue
        result=False
        break
    return result

#用户输出信息
```

```
def tprint(txt):
    g.ttmsg.insert(END, txt)
    g.ttmsg.see(END)

#用户输出信息，带颜色
def ttprint(txt,color):
    g.ttmsg.insert(END, txt,color)
    g.ttmsg.see(END)
```

下面是使用示例 9-6。

```
import HP_global as g
import HP_set
from HP_robot import *

#茉莉聊天机器人对话
q='北京'                                #问题
print('\n我: '+q)
huida='茉莉: '+moli_robot(q)           #回答
print(huida)

#图灵聊天机器人对话
q='广州天气'                            #问题
print('\n我: '+q)
huida='图灵: '+tuling_robot(q)         #回答
print(huida)

#小白对话系统
robot_init()
q='什么是股票'
print('\n我: '+q)
huida=g.ttrobot.answer(q,'simple')      #回答
print(huida)
```

程序运行结果:

我: 北京

茉莉: 北京是中华人民共和国的首都,中国的政治中心、科技和文化中心,是世界著名的古都和现代化国际城市。

我：广州天气

图灵：广州，周五 02 月 01 日，阴转小雨 无持续风向微风，最低气温 13 摄氏度，最高气温 19 摄氏度

我：什么是股票

股票是一种由股份有限公司签发的用以证明股东所持股份的凭证，它表明股票的持有者对股份公司的部分资本拥有所有权。由于股票包含经济利益，且可以上市流通转让，因此也是一种有价证券。

9.10 策略编辑器HP_edit

这节主要介绍 Python 的代码编辑运行模块，我们用 Frame 类来设计这个模块。我们主要完成两个功能的设计：一是工具条，二是代码编辑器。

HP_edit.py 模块中的完整代码请参照源代码。

下面见示例 9-7。

```
#-*- coding: utf-8 -*-
import tkinter as tk
import HP_global as g
import HP_set
from HP_edit import *
from HP_view import *

root = tk.Tk()
root.title('我的双代码编辑窗口')
setCenter(root,1200,800)

#划分为 2 个窗口
xxx=view2(root)
useredit(xxx.v[0]) #程序编辑框 1
useredit(xxx.v[1]) #程序编辑框 2
xxx.pack(fill=tk.BOTH, expand=1)
root.mainloop()
```

程序运行结果见图 9-6。



图 9-6 示例程序运行结果

9.11 总体框架构建模块HP_MainPage

这节我们将前面学习的知识进行汇总，构建一个小白量化投资系统的主框架。

1. 总框架的构成

小白量化投资系统的主框架包括主菜单、工具条、显示数据内容、显示股票 K 线图（包含输入日期、股票代码、指标的小工具）、回测面板（包含输入日期、股票代码、代码编辑窗和图形输出窗口）。

在此基础上，读者可利用提供的演示代码修改成为自己需要的量化平台。

框架的主要部分是定义为 g 空间的全局变量，所以用户程序能够接管或控制主框架本身。用户设计的新的控制面板或窗口，称为用户插件。同时，用户还可以通过框架来运行自己的回测程序、数据采集程序、自动下单程序，这些都被称为用户程序。

下面是小白量化投资系统主框架的布局 and 全局变量，如图 9-7 所示。

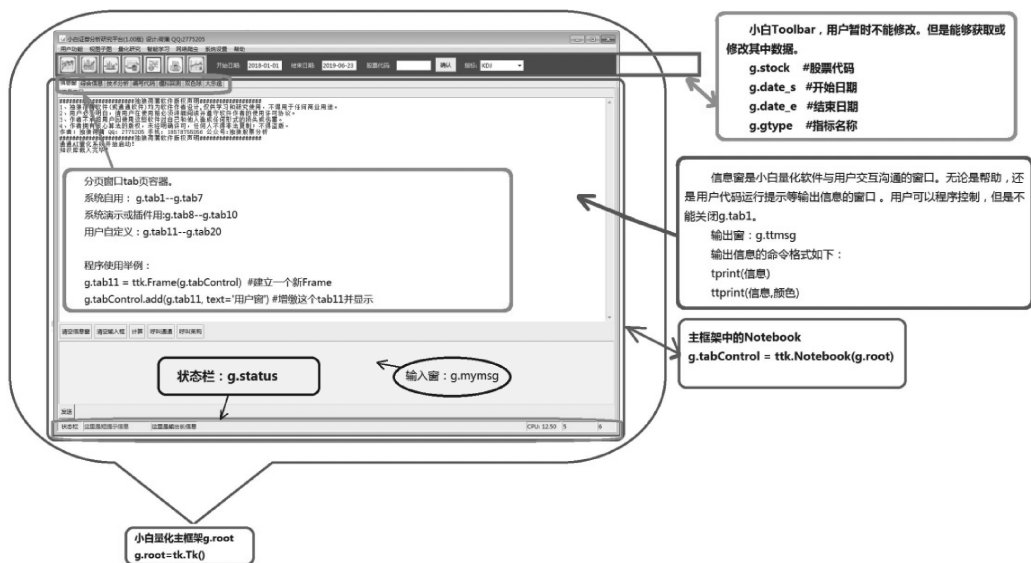


图 9-7 小白量化投资系统主框架的各控件对应的全局变量

2. 总框架的源代码

下面给出 HP_MainPage.py 的部分源代码。

```
#-*- coding: utf-8 -*-
import datetime as dt
from time import sleep
import tkinter as tk
import webbrowser
import os
import math
from math import *
from HP_view import * #菜单栏对应的各子页面
import HP_global as g
import HP_data as hp
from HP_edit import *
from HP_robot import *
from tkinter.filedialog import *
from tkinter.messagebox import *
from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg, NavigationToolbar2Tk) ##matplotlib 3.0.2
```

```

from PIL import Image, ImageTk
import threading

#小白量化软件主框架
class MainPage(object):
    def __init__(self, master=None):
        self.root = master #定义内部变量 root
        g.root=self.root
        self.tabControl=None
        self.w = g.winW
        self.h = g.winH
        self.root.title(g.title)
        self.staIco = g.ico
        self.root.geometry('%dx%d' % (self.w, self.h)) #设置窗口大小
        self.createUI()
        self.center()
        self.loop()

#生成界面
def createUI(self):
    self.createICO()
    self.createMenu2()
    self.createToolbar()
    g.status=StatusBar(self.root)
    g.status.pack(side=tk.BOTTOM, fill=tk.X)
    g.status.text(1, '这里是短提示信息') #在状态栏 2 输出信息
    g.status.text(2, '这里是输出长信息')
    g.status.config(1, color='red') #改变状态栏 2 信息的颜色
    g.status.config(5, width=5) #改变状态栏 6 的宽度
    g.status.set(3, 'CPU: %0.2f'%12.5) #格式输出信息
    self.createPage()

#主循环
def loop(self):
    self.root.resizable(True, True) #禁止修改窗口大小
    self.center() #窗口居中
    self.root.mainloop()

```

```

#退出
def _quit(self):
    #结束事件主循环,并销毁应用程序窗口
    self.root.quit()
    self.root.destroy()

#移动到窗口中间
def center(self):
    ws = self.root.winfo_screenwidth()
    hs = self.root.winfo_screenheight()
    x = int( (ws/2) - (self.w/2) )
    y = int( (hs/2) - (self.h/2) )
    self.root.geometry('{}x{}+{}+{}'.format(self.w, self.h, x, y))
    self.root.iconbitmap(self.staIco)

#打开网页函数
def web2(self):
    webbrowser.open("https://mp.weixin.qq.com/s/dAlD-S0JmfQ5kMQsludZEQ")

def web2a(self):
    webbrowser.open("https://blog.csdn.net/hepu8")

def web3a(self):
    webbrowser.open("https://book.yutiansut.com/")

def web3(self):
    webbrowser.open("http://tushare.org")

def web4(self):
    webbrowser.open("https://www.joinquant.com/help/api/help?name=JQData")

def web5(self):
    webbrowser.open("https://www.quantos.org/opendata/")

```

```

filename=''
#运行用户程序
def runUserfile(self):
    global filename
    filename = askopenfilename(defaultextension='.py')
    msg=''
    if filename == '':
        filename = ''
    else:
        f = open(filename,'r',encoding='utf-8',errors='ignore')
        msg=msg+f.read()
        f.close()

    try:
        print('开始运行用户代码。\\n')
        tprint('开始运行用户代码。\\n')
        exec(msg)
    except Exception as e:
        #print(type(e))
        tprint('用户代码出错:'+str(e)+'\\n','red')
        print('用户代码出错:'+str(e)+'\\n')
        showinfo(title='用户代码出错', message=str(e))

def mn1(self):
    try:
        filename='view/聚宽登录窗口.py'
        g.status.text(1,'')
        g.status.text(2,'聚宽登录')
        f = open(filename,'r',encoding='utf-8',errors='ignore')
        msg=f.read()
        f.close()
        exec(msg)
    except Exception as e:
        g.status.text(1,'')
        g.status.text(2,'用户代码出错:'+str(e))
        tprint('用户代码出错:'+str(e)+'\\n','red')
        print('用户代码出错:'+str(e)+'\\n')

```

```
def mn2(self):
    try:
        filename='view/二图画面 view2b.py'
        g.status.text(1, '')
        g.status.text(2, '')
        f = open(filename, 'r', encoding='utf-8', errors='ignore')
        msg=f.read()
        f.close()
        exec(msg)
    except Exception as e:
        g.status.text(1, '')
        g.status.text(2, '用户代码出错:'+str(e))
        ttprint('用户代码出错:'+str(e)+'\n', 'red')
        print('用户代码出错:'+str(e)+'\n')

def mn3(self):
    try:
        filename='view/三图画面 view3b.py'
        g.status.text(1, '')
        g.status.text(2, '')
        f = open(filename, 'r', encoding='utf-8', errors='ignore')
        msg=f.read()
        f.close()
        exec(msg)
    except Exception as e:
        g.status.text(1, '')
        g.status.text(2, '用户代码出错:'+str(e))
        ttprint('用户代码出错:'+str(e)+'\n', 'red')
        print('用户代码出错:'+str(e)+'\n')

def mn4(self):
    try:
        filename='view/四图画面 K 线.py'
        g.status.text(1, '')
        g.status.text(2, '')
```

```

        f = open(filename, 'r', encoding='utf-8', errors='ignore')
        msg=f.read()
        f.close()
        exec(msg)
    except Exception as e:
        g.status.text(1, '')
        g.status.text(2, '用户代码出错:'+str(e))
        ttprint('用户代码出错:'+str(e)+'\n', 'red')
        print('用户代码出错:'+str(e)+'\n')

def mn5(self):
    try:
        filename='view/运行用户程序窗口.py'
        g.status.text(1, '')
        g.status.text(2, '')
        f = open(filename, 'r', encoding='utf-8', errors='ignore')
        msg=f.read()
        f.close()
        exec(msg)
    except Exception as e:
        g.status.text(1, '')
        g.status.text(2, '用户代码出错:'+str(e))
        ttprint('用户代码出错:'+str(e)+'\n', 'red')
        print('用户代码出错:'+str(e)+'\n')

def mn6(self, fn=''):
    if fn=='':
        return
    try:
        filename='view/运行用户程序窗口.py'
        f = open(filename, 'r', encoding='utf-8', errors='ignore')
        msg=f.read()
        f.close()
        exec(msg)
        #time.sleep(3)
        g.UserView.openfile2(fn)
        g.UserView.runuc()

```



```

except Exception as e:
    g.status.text(1, '')
    g.status.text(2, '用户代码出错:'+str(e))
    ttprint('用户代码出错:'+str(e)+'\n', 'red')
    print('用户代码出错:'+str(e)+'\n')

def mn7(self, fn=''):
    if fn=='':
        return
    try:
        filename='view/运行用户程序窗口.py'
        f = open(filename, 'r', encoding='utf-8', errors='ignore')
        msg=f.read()
        f.close()
        exec(msg)
        #time.sleep(3)
        g.UserView.openfile2(fn)
        #g.UserView.runuc()
    except Exception as e:
        g.status.text(1, '')
        g.status.text(2, '用户代码出错:'+str(e))
        ttprint('用户代码出错:'+str(e)+'\n', 'red')
        print('用户代码出错:'+str(e)+'\n')

def mm1(self):
    g.status.text(1, '用户程序选股')
    g.status.text(2, '正在读入文件...')
    self.mn7('user/ts 选股.py')
    g.status.text(2, '')

def mm2(self):
    g.status.text(1, '网络爬虫')
    g.status.text(2, '正在获取网上股票新闻...')
    self.mn6('user/网络爬虫_股票新闻.py')
    g.status.text(2, '程序运行完成, 请查看结果! ')

def mm3(self):

```

```

g.status.text(1, '深度学习')
g.status.text(2, '正在读入文件...')
self.mn7('user/ts 深度学习程序.py')
g.status.text(2, '')

#创建菜单
def createMenu2(self):
    '''只支持两层嵌套'''
    menus = ['用户功能', '视图子图', '量化研究', '智能学习', '网络爬虫',
             '系统设置', '帮助']
    items = [['用户登录', '运行用户程序', '退出系统'], \
             ['信息窗', '综合信息', '代码窗', '回测窗', '分时图', '日线图',
              '二窗口演示', '三窗口演示', '四窗口演示'],
             ['选股策略', '单只股买卖策略', '多只股票买卖策略'], \
             ['智能学习'], \
             ['股票新闻'], \
             ['系统设置'], \
             ['荷蒲量化', '聚宽量化', 'QUANTAXIS 量化', 'tushare 网',
              'OpenDataTools']]
    callbacks = [[None, self.runUserfile, self._quit], \
                 [self.bt1, self.bt2, self.bt4, self.bt5, self.bt9, self.bt3, self.mn2, self.
mn3, self.mn4],
                 [self.mm1, self.bt5, None],
                 [self.mm3], [self.mm2], [None], [self.web2a, self.web4,
self.web3a, self.web3, self.web5]]
    icos = [[self.img1, self.img4, self.img2], \
            [None, None, None, None, None, None, None, None, None],
            [None, None, None],
            [None], [None], [None], [self.img2, None, None, None, None]]

    menubar = Menu(self.root)
    for i, x in enumerate(menus):
        m = Menu(menubar, tearoff=0)
        for item, callback, ico in zip(items[i], callbacks[i], icos[i]):

            if isinstance(item, list):
                sm = Menu(menubar, tearoff=0)

```

```

        for subitem, subcallback, subico in zip(item[1:],
callback, ico):

            if subitem == '-':
                sm.add_separator()
            else:
                sm.add_command(label=subitem, command=subcallback,
image=subico, compound='left')
            m.add_cascade(label=item[0], menu=sm)
        elif item == '-':
            m.add_separator()
        else:
            m.add_command(label=item, command=callback, image=ico,
compound='left')
        menubar.add_cascade(label=x, menu=m)
    self.root.config(menu=menubar)

#创建菜单
def createMenu(self):
    '''只支持两层嵌套'''
    menus = ['用户功能','数据下载','量化研究','智能学习','历史回测','网络爬虫','系统设置','帮助']
    items = [['用户注册','运行用户程序','退出系统'],['指数数据下载','股票数据下载'],
            [],[],[],[],[],['通通量化','tushare 网','聚宽量化','OpenDataTools']]
    callbacks = [[None, self.runUserfile,self._quit],[None,None],
            [],
            [],[],[],[],[self.web2a,self.web3,self.web4,self.web5]]
    icos = [[self.img1,self.img4, self.img2],[None,None],
            [],
            [],[],[],[],[self.img2,None,None,None,None]]

    menubar = Menu(self.root)
    for i,x in enumerate(menus):
        m = Menu(menubar, tearoff=0)
        for item, callback, ico in zip(items[i], callbacks[i], icos[i]):

```

```

        if isinstance(item, list):
            sm = Menu(menubar, tearoff=0)
            for subitem, subcallback, subico in zip(item[1:],
callback, ico):

                if subitem == '-':
                    sm.add_separator()
                else:
                    sm.add_command(label=subitem, command=
subcallback, image=subico, compound='left')
            m.add_cascade(label=item[0], menu=sm)
        elif item == '-':
            m.add_separator()
        else:
            m.add_command(label=item, command=callback, image=ico,
compound='left')

        menubar.add_cascade(label=x, menu=m)
        self.root.config(menu=menubar)

#生成所有需要的图标
def createICO(self):
    self.img1 = ImageTk.PhotoImage(Image.open('./t4.jpg'))
    self.img2 = ImageTk.PhotoImage(Image.open('./t1.jpg'))
    self.img3 = ImageTk.PhotoImage(Image.open('./t2.jpg'))
    self.img4 = ImageTk.PhotoImage(Image.open('./t3.jpg'))
    self.png1 = ImageTk.PhotoImage(Image.open('tt_zs.png'))
    self.png2 = ImageTk.PhotoImage(Image.open('tt_jjhq.png'))
    self.png3 = ImageTk.PhotoImage(Image.open('tt_ph.png'))
    self.png4 = ImageTk.PhotoImage(Image.open('tt_bb.png'))
    self.png5 = ImageTk.PhotoImage(Image.open('tt_mg.png'))
    self.png6 = ImageTk.PhotoImage(Image.open('tt_wd.png'))
    self.png7 = ImageTk.PhotoImage(Image.open('tt_hq.png'))

def bt1(self):
    self.tabControl.select(g.tab1)

def bt2(self):

```

```

        self.tabControl.select(g.tab2)

def bt3(self):
    self.tabControl.select(g.tab3)

def bt4(self):
    self.tabControl.select(g.tab4)

def bt5(self):
    self.tabControl.select(g.tab5)

def bt6(self):
    self.tabControl.select(g.tab6)

def bt7(self):
    self.tabControl.select(g.tab7)

def bt8(self):
    self.tabControl.select(g.tab8)

def bt9(self):
    self.tabControl.select(g.tab9)

#生成工具条
def createToolbar(self):
    toolframe =tk.Frame(self.root, height=20,bg='#1E488F')
    frame = tk.Frame(toolframe, bg='#1E488F')
    ttk.Button(frame, width=20, image=self.png1,
command=self.bt1).grid(row=0, column=0, padx=1, pady=1, sticky=tk.E)
    ttk.Button(frame, width=20, image=self.png2,
command=self.bt2).grid(row=0, column=1, padx=1, pady=1, sticky=tk.E)
    ttk.Button(frame, width=20, image=self.png3,
command=self.bt3).grid(row=0, column=2, padx=1, pady=1, sticky=tk.E)
    ttk.Button(frame, width=20, image=self.png4,
command=self.bt4).grid(row=0, column=3, padx=1, pady=1, sticky=tk.E)
    ttk.Button(frame, width=20, image=self.png5,
command=self.bt5).grid(row=0, column=4, padx=1, pady=1,sticky=tk.E)

```

```

        ttk.Button(frame, width=20, image=self.png6,
command=self.bt6).grid(row=0, column=5, padx=1, pady=1,sticky=tk.E)
        ttk.Button(frame, width=20, image=self.png7,
command=self.bt7).grid(row=0, column=6, padx=1, pady=1,sticky=tk.E)

        Label(frame, text=' ',bg='#1E488F').grid(row=0, column=7)
        label1 = Label(frame, width=10, text='开始日期:
',ancho=S,bg='#1E488F',fg='white')
        label1.grid(row=0, column=8)
        #输入框 (Entry)
        date_s = StringVar()
        entrydates = Entry(frame, width=10, textvariable=date_s)
        entrydates.grid(row=0, column=9)
        date_s.set(g.sday)
        Label(frame , text=' ',bg='#1E488F').grid(row=0, column=10)
        label2 = Label(frame ,width=10, text='结束日期: ',bg='#1E488F',
fg='white')
        label2.grid(row=0, column=11)
        #输入框 (Entry)
        date_e = StringVar()
        entrydateee = Entry(frame, width=10,textvariable=date_e)
        nowdt=dt.datetime.now()           #获取今天日期
        g.eday=nowdt.strftime('%Y-%m-%d')  #将日期格式转字符串格式
        date_e.set(g.eday)
        entrydateee.grid(row=0, column=12)
        Label(frame , text=' ',bg='#1E488F').grid(row=0, column=13)
        label3 = Label(frame ,width=10, text='股票代码:',bg='#1E488F',
fg='white')
        label3.grid(row=0, column=14)
        #输入框 (Entry)
        stock = StringVar()
        stock.set(' ')
        entrystock = Entry(frame,width=10, textvariable=stock)
        entrystock.grid(row=0, column=15)

        #确定按钮
        def rtnkey(event=None):

```

```

ds=date_s.get()
de=date_e.get()
stockn=stock.get()
stockn=stockn.strip()
stockn=stockn.zfill(6)
g.gtype=book.get()
g.stock=stockn
g.sday=ds.strip()
g.eday=de.strip()
df2=hp.get_k_data(stockn,ktype='D',start=ds,end=de,index=
False,autype='qfq')
g.df=df2.copy()
df3=hp.tstojq(df2)
if g.plotPage !=None:
    g.plotPage.canvas._tkcanvas.pack_forget()
    g.plotPage.pack_forget()
    g.plotPage=None

g.plotPage = plotFrame(g.tab3,df3,stockn,g.gtype)
g.plotPage.pack(fill=X)
self.tabControl.select(self.tab3)

#确定按钮
def rtnkey(event=None):
    ds=date_s.get()
    de=date_e.get()
    stockn=stock.get()
    stockn=stockn.strip()
    stockn=stockn.zfill(6)
    g.gtype=book.get()
    g.stock=stockn
    g.sday=ds.strip()
    g.eday=de.strip()
    df2=hp.get_k_data(g.stock,ktype='D',start=g.sday,end=g.eday,
index=False,autype='qfq')
    if g.tab3!=None:
        g.tabControl.forget(g.tab3)

```

```

g.tab3=None

#用户自建新画面
g.tab3 = tk.Frame(g.tabControl)
g.tabControl.add(g.tab3, text='日线图')
self.tab3=g.tab3
self.tabControl.select(self.tab3)
axview3x(self.tab3,df2,t=g.stock,n=2,f1='VOL',f2=g.gtype)
self.tabControl.select(self.tab3)

#绑定回车键
entrystock.bind('<Key-Return>', rtnkey)
Label(frame, text=' ',bg='#1E488F').grid(row=0, column=16)
#按钮 (Button)
getname = Button(frame , width=5,text='确认' ,command=rtnkey)
getname.grid(row=0, column=17)
Label(frame , text=' ',bg='#1E488F').grid(row=0, column=18)
label4 = Label(frame,width=5,text='指标:',bg='#1E488F',fg='white')
label4.grid(row=0, column=19)
#增加 Combobox
book = tk.StringVar()
bookChosen = ttk.Combobox(frame, width=10, textvariable=book)
bookChosen['values'] = ('KDJ', 'MACD', 'RSI', 'OBV', 'BOLL', '自定义',
                        'HPYYX')
bookChosen.grid(row=0, column=20)
bookChosen.current(0) #设置初始显示值, 值为元组['values']的下标
bookChosen.config(state='readonly') #设为只读模式
frame.pack(side=tk.LEFT)
toolframe.pack(fill=tk.X)

#回测面板
def hc(self,frame) :
    Label(frame, text=' ').grid(row=0, column=0)
    label1 = Label(frame, width=10, text='开始日期: ', ancho=S)
    label1.grid(row=0, column=1)
    #输入框 (Entry)
    g.hcdate_s = StringVar()

```



```

entrydates = Entry(frame, width=10, textvariable=g.hcdate_s)
entrydates.grid(row=0, column=2)
g.hcdate_s.set(g.sday)
Label(frame, text=' ').grid(row=0, column=3)
label2 = Label(frame, width=10, text='结束日期: ')
label2.grid(row=0, column=4)
#输入框 (Entry)
g.hcdate_e = StringVar()
entrydatee = Entry(frame, width=10, textvariable=g.hcdate_e)
g.hcdate_e.set(g.eday)
entrydatee.grid(row=0, column=5)
Label(frame, text=' ').grid(row=0, column=6)
label3 = Label(frame, width=10, text='初始资金(元):')
label3.grid(row=0, column=7)
#输入框 (Entry)
g.hczj = StringVar()
g.hczj.set(g.money)
entryzj = Entry(frame, width=12, textvariable=g.hczj)
entryzj.grid(row=0, column=8)
Label(frame, text=' ').grid(row=0, column=6)
label4 = Label(frame, width=12, text='止损幅度(%):')
label4.grid(row=0, column=9)
#输入框 (Entry)
g.hczs = StringVar()
g.hczs.set(g.stop_loss_range)
entryzs = Entry(frame, width=10, textvariable=g.hczs)
entryzs.grid(row=0, column=10)
Label(frame, text=' ').grid(row=0, column=11)
label4 = Label(frame, width=12, text='股票代码:')
label4.grid(row=0, column=12)
#输入框 (Entry)
g.hcstock = StringVar()
g.hcstock.set('000001')
entryst = Entry(frame, width=10, textvariable=g.hcstock)
entryst.grid(row=0, column=13)
Label(frame, text=' ').grid(row=0, column=14)
def stt():

```

```

        if g.UserCanvas!=None:
            g.UserPlot.cla()
            g.UserPlot.close()
            g.UserCanvas._tkcanvas.pack_forget()
            g.UserCanvas=None
        if g.UserPlot !=None:
            g.UserPlot.pack_forget()
    #按钮 (Button)
    getname = Button(frame , text='清除画面' ,command=stt)
    getname.grid(row=0, column=15)

#量化软件主页面
def createPage(self):
    self.tabControl = ttk.Notebook(self.root) #Create Tab Control
    self.tab1 = ttk.Frame(self.tabControl) #Add a third tab
    self.tabControl.add(self.tab1, text='信息窗') #Make second tab visible
    self.tab2 = ttk.Frame(self.tabControl) #Create a tab
    self.tabControl.add(self.tab2, text='综合信息') #Add the tab
    self.tab3 = ttk.Frame(self.tabControl) #Add a second tab
    self.tabControl.add(self.tab3, text='技术分析') #Make second tab visible
    self.tab4 = ttk.Frame(self.tabControl) #Add a third tab
    self.tabControl.add(self.tab4, text='编写代码') #Make second tab visible
    self.tab5 = ttk.Frame(self.tabControl) #Add a third tab
    self.tabControl.add(self.tab5, text='模拟回测') #Make second tab visible
    self.tab6 = ttk.Frame(self.tabControl) #Add a third tab
    self.tabControl.add(self.tab6, text='双色球') #Make second tab visible
    self.tab7 = ttk.Frame(self.tabControl) #Add a third tab
    self.tabControl.add(self.tab7, text='大乐透') #Make second tab visible
    self.tabControl.pack(expand=1, fill="both") #Pack to make visible
    g.tabControl=self.tabControl
    g.tab1=self.tab1
    g.tab2=self.tab2
    g.tab3=self.tab3
    g.tab4=self.tab4
    g.tab5=self.tab5
    g.tab6=self.tab6
    g.tab7=self.tab7

```

```

talkfrm = ttk.LabelFrame(self.tab1, text='信息窗口')
talkfrm.pack(side=tk.TOP, fill=tk.BOTH, expand=1)
talk=mytalk(talkfrm)
g.ttmsg.insert(END, g.tttext, 'blue')
df1=hp.get_today_all()
g.tttree=mygrid(self.tab2,df1)
df2=hp.get_k_data('000001',ktype='D',start='2018-04-01',end=
'2019-02-01',index=False,autype='qfq')
axview3x(self.tab3,df2,t='000001',n=2,f1='VOL',f2='HPYYX')
myed=useredit(self.tab4)

#-----Tab2 控件介绍-----#
monty2 = ttk.LabelFrame(self.tab5, text='回测设置')
monty2.pack()
myhc=self.hc(monty2)

#创建 frame 容器
self.frmA = ttk.LabelFrame(self.tab5, text='回测输出画板')
self.frmA.rowconfigure(0,weight=1)
self.frmA.columnconfigure(0,weight=1)
self.frmA.pack(side=tk.TOP, fill=tk.BOTH, expand=1)
self.frmL = Frame(self.frmA,width = g.winW-600,height = 800,
relief=tk.SUNKEN)
self.frmR = Frame(self.frmA,width = 600,height = 800,relief=
tk.SUNKEN)
g.UserFrame=self.frmL

#窗口布局
self.frmL.grid(row = 0, column = 0,sticky=tk.NSEW)
self.frmR.grid(row = 0, column = 1,sticky=tk.NSEW)

myed2=myedit(self.frmR)

ssq=hp.get_ssq()
mygrid(self.tab6,ssq)
dlt=hp.get_dlt()
mygrid(self.tab7,dlt)

```

9.12 主程序模块HP_main

下面为量化程序的主程序代码：

```
import threading
import HP_global as g
import HP_set
from HP_MainPage import *
from HP_view import *
from PIL import Image, ImageTk
from tkinter import *
import threading
from HP_robot import *
if __name__ == "__main__":
    from PIL import Image, ImageTk
    g.root = tk.Tk()
    g.root.title(g.title)
    g.login=True
    window1 = tk.IntVar(g.root, value=0)
    window1.set(1)
    photo=ImageTk.PhotoImage(Image.open('tt_welcome2.jpg'))
    w=photo.width()
    h=photo.height()
    w1 = myWindow(g.root,g.title,w,h)
    showIco(w1.top,g.ico)
    label=tk.Label(w1.top,image=photo)
    label.grid(row=0, column=0, padx=1, pady=1, sticky=tk.E)
    setCenter(w1.top,photo.width()+4,photo.height()+4)
    reSizable(w1.top,False, False)
    def fun_timer():
        global timer
        w1.destroy()
        timer.cancel()
    timer = threading.Timer(15, fun_timer)
    timer.start()
```

```
window1.set(0)
g.ttext= '小白 AI 量化系统开始启动! \n'
robot_init()
g.mainform=MainPage(g.root)
```

程序的运行结果见图 9-8。

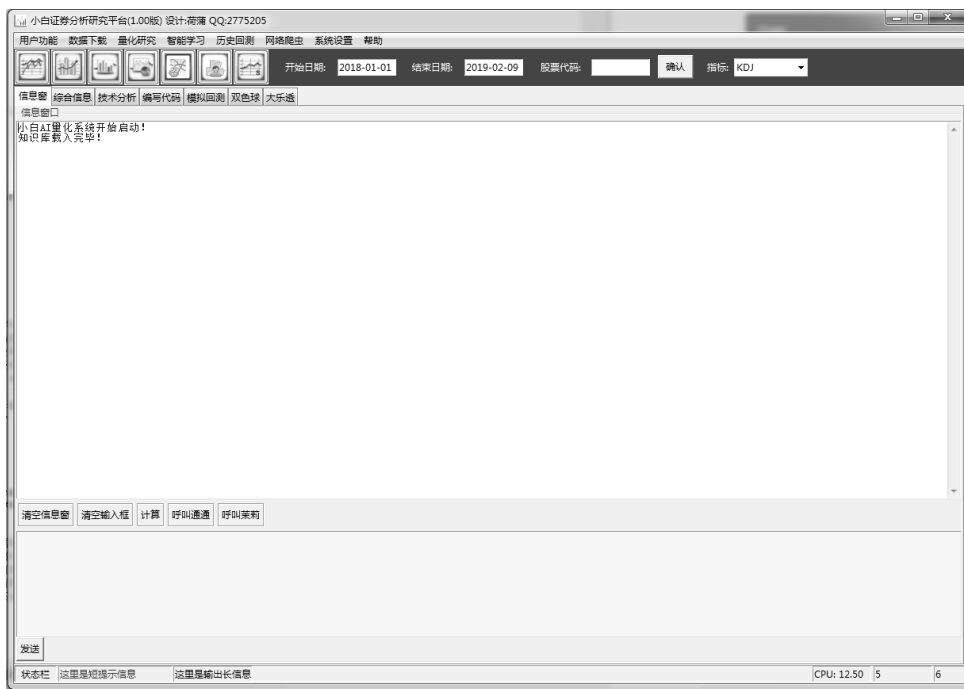


图 9-8 示例程序运行结果

10

第 10 章

分析回测与预测

本章介绍在 Python 中实现股票分析回测与预测的方法，这对于不熟悉投资分析的读者来说能起到抛砖引玉的作用。

10.1 投资分析方法

目前，投资品种比较多，我们很难能够快速扫描、分析全部品种，建议按照下面几个步骤进行分析。

- (1) 选股。在盘后，建立一个备选股票池。
- (2) 择时买入。在盘中，分析股票池中具有买入时机的股票。
- (3) 持仓分析。分析持仓股票是否有卖出信号，是否要止损或者止盈。
- (4) 如果在开盘期间，就重复 (2)，直到闭市。

10.2 选股

选股有多种方式，传统方式有基本面选股和热点板块选股，目前还有量化分析选股、深度学习选股等。

这里我们主要介绍基本面选股和热点板块选股。



1. 基本面选股

我们关注的基本面参数主要有市盈率、市净率、流通盘。

股票内在价值的核心由每股资产净值和每股收益率两部分组成。

如果股票净资产是真实的,那么 10 元每股的净资产其股价怎么也不会跌到 5 元,而折价到 8 元或 9 元是可能存在的,这时可能很快就被投资者买回到 10 元以上。为了能够体现股价和净资产之间的关系,就需要考虑市净率。市净率=股价/每股净资产。

每股收益是每股股票产生的纯利润。假如股票收益数据是真实的,且每年都会把收益分红给投资者。如果股价为 10 元,每年分红 1 元,那么投资这只股票 10 年就会收回本金。即使不给投资者分红,股票净资产也会每年增长 1 元,但是 10 元股票每股收益 1 元与 100 元股票每股收益 1 元的本金回收效果是不同的。为了能够衡量回本效率,就需要参考市盈率。

为了能够选择持续增长的股票,就要分析上市公司连续几年的收益情况,这就需要利用量化分析的手段来实现。

一般流通盘(也称为流通股本、流通股份)比较小的股票比较活跃,波动也较大。流通盘大的股票,因股票总市值较高,市场流动资金很难拉升股价,所以一般不受投资者的喜爱。一般流通股本在 1 亿以上的个股称为大盘股;5000 万至 1 亿的个股称为中盘股;不到 5000 万规模的称为小盘股,活跃涨幅较大的都是中小盘股。

ST 属于快要退市的股票,一般市盈率和市净率都比较差,多数为负值。

对于热点的股票,一般根据所属板块或所属指数成分股来选择。

下面我们采用 Tushare 股票数据做一个基本面选股的示例。

Tushare 的 `ts.get_today_all()`方法能获取沪深上市公司的实时数据,数据内容见表 10-1。

表 10-1 `ts.get_today_all()`方法获取的数据

| 数据代码 | 含义 |
|---------------|------|
| code | 股票代码 |
| name | 股票名称 |
| changepercent | 涨跌幅 |
| trade | 现价 |
| open | 开盘价 |

续表

| 数据代码 | 含义 |
|---------------|-------|
| high | 最高价 |
| low | 最低价 |
| settlement | 昨日收盘价 |
| volume | 成交量 |
| turnoverratio | 换手率 |
| amount | 成交金额 |
| per | 市盈率 |
| pb | 市净率 |
| mktcap | 市值 |
| nmc | 流通市值 |

选股流程如下：

(1) 获取沪深上市公司的实时数据，其中包含股票代码、市盈率、市净率。其中，删除业绩较差的 ST 股票、业绩亏算的股票和净资产为负的股票。

(2) 选取市盈率前 100 名的股票。

(3) 选取市净率前 100 名的股票。

(4) 对所选取的股票代码进行运算，选取满足条件的股票代码保存到板块 zxgl.dat 文件中。

(5) 读取文件 zxgl.dat 中的数据，并还原为股票池列表数据。

程序代码见示例 10-1。

```
#通过基本面选股，建立股票池
import pickle
import tushare as ts
import HP_data as hp
#获取最新股票数据
df=ts.get_today_all()
df1=df.copy()                                #建立一个备份
n=100                                         #选择前 n 个数据
#删除业绩较差的 ST 股票
df1['a']=[('ST' in x) for x in df1.name.astype(str)] #先给 ST 股票做标记 a
df1=df1.set_index('a')                      #将 a 设置为索引
df1=df1.drop(index=[True])                  #删除 ST 股票
df1=df1.reset_index(drop=True)              #重建默认索引
```



```

#删除业绩亏损的股票
df1=df1[df1.per >0]
#删除净资产为负的股票
df1=df1[df1.pb >0]
#选取市盈率前 100 名的股票
df2=df1.sort_values(by=['per'],ascending=True).head(n)
#选取市净率前 100 名的股票
df3=df1.sort_values(by=['pb'],ascending=True).head(n)
#生成股票代码集合, 进行集合运算
g2=set(df2.code)                #低市盈率股票代码
g3=set(df3.code)                #低市净率股票代码
g=g2&g3                          #集合交运算
zxg1=list(g)                     #把集合转为列表
print()
print('原始筛选结果: ',zxg1)
#把结果保存到 zxg1.dat 文件中
f = open('zxg1.dat', 'wb')
pt=pickle.dumps(zxg1,0)
f.write(pt)
f.close()
#获取 zxg1.dat 文件, 并还原为股票池数据
f = open('zxg1.dat', 'rb')
zxg2=pickle.load(f)
f.close()
print('板块文件中数据: ',zxg2)

```

程序运行结果:

```

[Getting data:]#####
原始筛选结果: ['600823', '000587', '000898', '600525', '600051', '601666',
'600000', '601988', '000402', '600708', '600657', '601818', '000780',
'600240', '601328', '000059', '600015', '601166', '601169', '000980',
'600016', '600919', '601998', '000625', '002354']
板块文件数据: ['600823', '000587', '000898', '600525', '600051', '601666',
'600000', '601988', '000402', '600708', '600657', '601818', '000780',
'600240', '601328', '000059', '600015', '601166', '601169', '000980',
'600016', '600919', '601998', '000625', '002354']

```

2. 热点板块选股

热点板块是当前市场上交投比较活跃、成交量比较大、板块内个股活跃度高于市场平均水平的行业或地域板块。

Tushare 的 `ts.get_stock_basics()` 方法可以获取沪深上市公司的基本情况，以此获取流通股本、所属行业和地区的信息，数据内容见表 10-2。

表 10-2 `ts.get_stock_basics()` 方法获得的数据

| 数据代码 | 含义 |
|------------------|----------|
| code | 股票代码 |
| name | 股票名称 |
| industry | 所属行业 |
| area | 地区 |
| pe | 市盈率 |
| outstanding | 流通股本（亿元） |
| totals | 总股本（亿元） |
| totalAssets | 总资产（万元） |
| liquidAssets | 流动资产 |
| fixedAssets | 固定资产 |
| reserved | 公积金 |
| reservedPerShare | 每股公积金 |
| esp | 每股收益 |
| bvps | 每股净资 |
| pb | 市净率 |
| timeToMarket | 上市日期 |
| undp | 未分利润 |
| perundp | 每股未分配 |
| rev | 收入同比（%） |
| profit | 利润同比（%） |
| gpr | 毛利率（%） |
| npr | 净利润率（%） |
| holders | 股东人数 |

一般，牛市成交量放大交投活跃，熊市成交量萎缩交投冷清。因此，牛市行情对券商板块的业绩增长有绝对的影响。在不能把握其他热点板块时，我们可以选择

流通盘在券商板块中相对较小的股票放到备选股票池。

选股流程如下：

(1) 获取沪深上市公司基本情况和所属行业等方面的数据，这里我们选择券商行业。

(2) 选取流通盘小于 30 亿股的券商股。

(3) 保存为股票池并放到 `zxc2.dat` 文件中。

获取股票市盈率和市净率的代码见示例 10-2。

```
#通过基本面选股，建立股票池
import pickle
import tushare as ts
import HP_data as hp
#获取股票基本信息
base=ts.get_stock_basics()
#选取券商板块股
df4=base[base.industry=='证券']
#选取流通盘小于 30 亿股
df4=df4[df4.outstanding<=30]
#生成股票代码集合，进行集合运算
g=list(df4.index)          #低市盈率股票代码
print('g= ',g)
#把结果保存到文件中
f = open('zxc2.dat', 'wb')
pt=pickle.dumps(g,0)
f.write(pt)
f.close()
print('pt= ',pt)
#获取自选股文件
f = open('zxc1.dat', 'rb')
zxc2=pickle.load(f)
f.close()
print('zxc2= ',zxc2)
```

程序运行结果：

```
g= ['601375', '600621', '002797', '601162', '601990', '601878',
'601555', '600909', '000728', '601066', '002939', '601108', '600864',
```

```
'601881', '600155', '000712', '002670', '002926', '601198', '000686',
'002500', '002945']

pt= b'(lp0\nV601375\np1\naV600621\np2\naV002797\np3\naV601162\np4\n
aV601990\np5\naV601878\np6\naV601555\np7\naV600909\np8\naV000728\np9\na
V601066\np10\naV002939\np11\naV601108\np12\naV600864\np13\naV601881\np14
\naV600155\np15\naV000712\np16\naV002670\np17\naV002926\np18\naV601198\n
p19\naV000686\np20\naV002500\np21\naV002945\np22\na.'
```

```
zxcg2= ['600823', '000587', '000898', '600525', '600051', '601666',
'600000', '601988', '000402', '600708', '600657', '601818', '000780',
'600240', '601328', '000059', '600015', '601166', '601169', '000980',
'600016', '600919', '601998', '000625', '002354']
```

10.3 择时买入

择时买入的信号分为两种情况：一是强势追涨，二是弱势抄底。

强势追涨一般用于熊市中的强势股票或者在牛市行情中操作；弱势抄底是在股票由弱开始走强的瞬间买入股票。

1. 追涨买点信号

很多人喜欢用双均线买点方法，即在 5 日收盘价均线上穿 20 日收盘价均线时买入，但这在回测时成功率却不高。这是为什么呢？这主要是因为完全没有理解这个方法的含义。

双均线买点方法是牛市追涨买点进行操作的，因此，在熊市中操作的失败概率很高。

股票价格具有压缩的特性，压缩到极限就会产生膨胀动力，在均线系统上也有相同的压缩表现。

均线的含义，假定收盘价近似代表一天交易双方的买卖成本价，由于买卖是同时发生的，即买入者和卖出者同时操作，因此买卖成对出现。卖出者可能不再持有该股，换成买入者持有该股，因此只有买入者有成本价。那么，5 日收盘价均线就代表 5 日内买入该股的成本价，120 日收盘价均线就代表 120 日内买入该股的成本价。我们可以想象一下，5 日、10 日、20 日、60 日、120 日、250 日的收盘价均线都在一起，它们是不是与一年来短期和长期买入者的成本价很接近，是不是感觉长

短均线都被压缩到了一起。这时只要股价剧烈波动就会引起暴跌或暴涨，投资者要提防。

为什么会这样？我们可以把均线近似看作成本线，比如 5 日均线就是 5 日成本线、120 日均线就是 120 日成本线。当 K 线被压缩时，上涨和下跌的空间急剧减少，因此短期成本和长期成本十分接近，这时可以认为市场上没有多少人获利。因为每只股票都有主力，这时股价也非常接近主力的成本价，所以一旦有股价上涨的机会，主力就会急速拉升使股价脱离自己的成本区。

我们可以从下面的图形中看到这种 K 线被压缩后的走势，如图 10-1 所示。



图 10-1 “600677 航天通讯”日 K 线及均线图

在这种压缩状态下，5 日均线上穿 20 日均线多数情况会获利。反之，如果持有该股，就要及时出局。

股票价格除了具有压缩特性外，还具有共振特性。

回顾历史走势，可以发现：股票走势经常大起大落，即一旦从低位启动产生向

上突破，股价就如脱缰的野马奔腾向前；而一旦从高位产生向下突破，股价又如决堤的江水一泻千里。这就是共振作用在股市中的反映。

共振可以产生势，而这种势一旦产生，向上向下的威力都极大。它能引发人们的情绪和操作行为，产生一边倒的情况。在向上时，人们情绪高昂，蜂拥入市；在向下时，人人恐慌，股价狂泻，就如同遇到“世界末日”，威廉·江恩称之为价格崩溃。

共振特性在均线系统上的表现为至少 3 根长短不同的均线在两个周期内都发生了交叉。例如，同一周期内 5 日均线“MA5”上穿 10 日均线“MA10”，同时“MA10”上穿 20 日均线“MA20”，这看起来像一张蜘蛛网。上穿蜘蛛预示股价要上涨，我们称为“金蜘蛛”；下穿蜘蛛预示股价要下跌，我们称为“黑蜘蛛”。如图 10-2 中的“601700 风范股份”日 K 线图所示，在 2018 年 11 月 26 日，5 日均线、10 均线、60 日均线相聚到一点，并且交叉，这就是金蜘蛛。

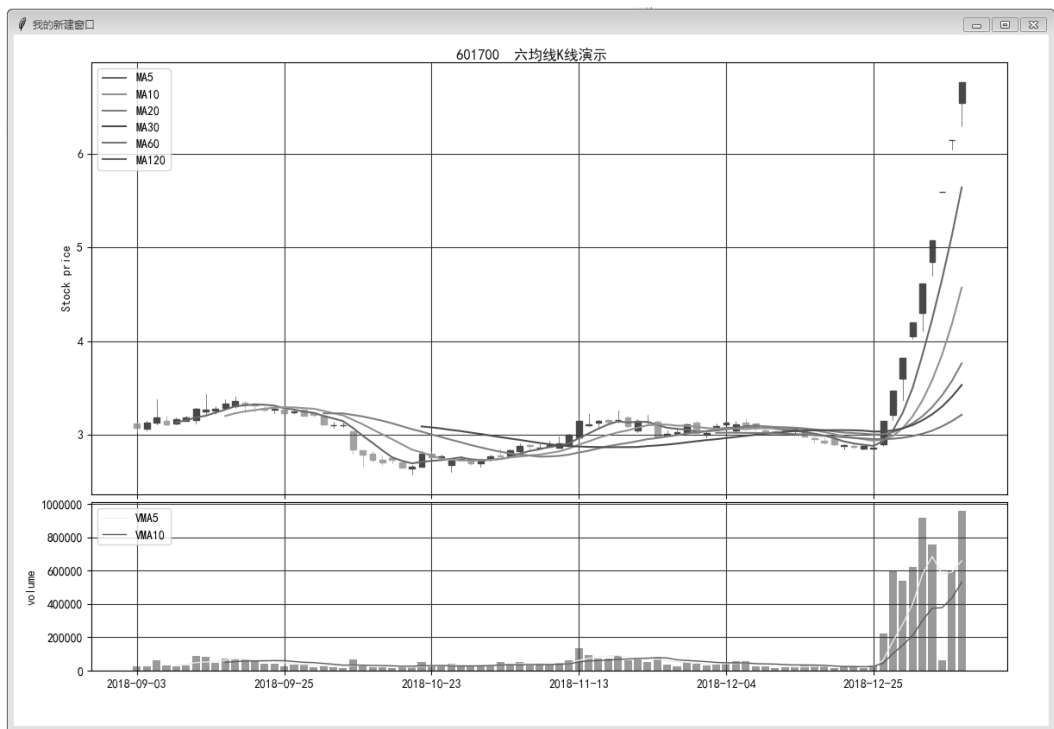


图 10-2 “601700 风范股份”日 K 线图

通过上面的讲解，读者就很容易明白均线指标的意义了。

仅仅均线符合要求还不够，因为买卖的同时伴随着成交量的变化，所以成交量的变化是确认成交价分析对错的依据。一般情况下，成交量随着股价的上涨而增加，随着股价的下跌而降低。所以，当我们以均线作依据买入股票时，一定要预判当日的成交量是否大于前一日的成交量。

见示例 10-3。

```
#通过基本面选股，建立股票池
import matplotlib
import matplotlib.pyplot as plt
import pickle
import tushare as ts
import HP_global as g           #全局变量命名空间
import HP_set                   #初始化系统全局变量
from HP_formula import *       #引入公式基本函数
#我们设定股票池

zxcg=['601988','601328','601818','600015','601166','600909','601700']
##回测程序开始
date_s='2017-01-01'           #回测开始时间
date_e='2019-01-04'           #回测结束时间
plate=zxcg                     #股票池
Acode=[]                      #保存股票代码
Asdf = []                     #保存股票数据
i=0
lmax=0
lendata=0
for code in plate:
    print(i,code)
    data = ts.get_k_data(code,ktype='D',start=date_s,end=date_e)
    lendata=len(data)
    if lendata<200:             #数据少于 200 天
        continue
    if lmax<lendata:
        lmax=lendata
        i+=1
    CLOSE=data['close']
    OPEN=data['open']
    VOL=data['volume']
```

```

data['MA5']=MA(CLOSE,5)
data['MA10']=MA(CLOSE,10)
data['MA20']=MA(CLOSE,20)
data['MA30']=MA(CLOSE,30)
data['MA60']=MA(CLOSE,60)
data['MA120']=MA(CLOSE,120)
data['MA250']=MA(CLOSE,250)
data['B1']=CROSS(data['MA5'],data['MA20'])      #5 日上穿 20 日
data['B2']=CROSS(data['MA5'],data['MA60'])      #5 日上穿 60 日
data['B3']=CROSS(data['MA5'],data['MA120'])     #5 日上穿 120 日
data['B4']=CROSS(data['MA20'],data['MA60'])     #20 日上穿 60 日
data['B5']=CROSS(data['MA20'],data['MA120'])    #20 日上穿 120 日
data['V1']=VOL/REF(VOL,1)      #今日 VOL 与昨日 VOL 比较
data['MAXX']=data[['close','MA5','MA10','MA20','MA30','MA60',
'MA120','MA250']].max(1)
data['MINX']=data[['close','MA5','MA10','MA20','MA30','MA60',
'MA120','MA250']].min(1)
data['TWO']=2
data['ZERO']=0
data['B6']=data['B1']+data['B2']+data['B3']+data['B4']+data['B5']
#交叉信号叠加
data['B7']=SUM(data['B6'],2)      #判断 2 日的买点信号
data['BB']=IF((CLOSE>OPEN)&(data['B7']>=2) &(data['V1']>=1.50)
&(((data['MAXX']-data['MINX'])/CLOSE)<0.5) ,data['TWO'],data['ZERO'])
Acode.append(code)
Asdf.append(data)

def MA5x20(i):                    #绘图函数
    plt.figure()
    plt.title(Acode[i])
    Asdf[i].MA5.plot.line(legend=True)
    Asdf[i].MA10.plot.line(legend=True)
    Asdf[i].MA20.plot.line(legend=True)
    Asdf[i].MA30.plot.line(legend=True)
    Asdf[i].MA60.plot.line(legend=True)
    Asdf[i].MA120.plot.line(legend=True)
    Asdf[i].BB.plot.line(legend=True)
    plt.show()

MA5x20(6)                        #显示列表 6 数据

```


程序运行结果见图 10-3。

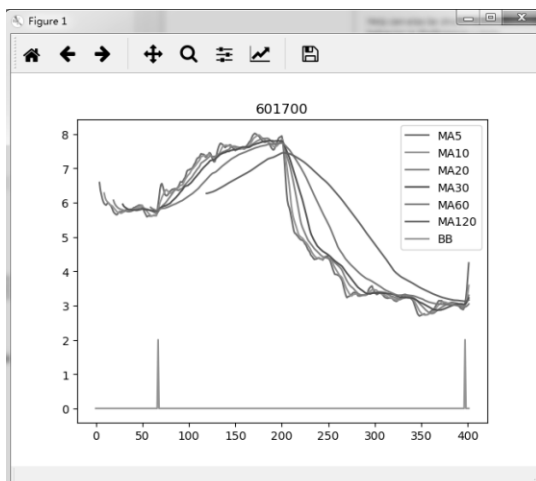


图 10-3 运行结果

2. 抄底买入信号

抄底买入信号一般发生在股市走弱时。这时我们不是在看到股票到底时去买，因为下跌趋势的股票无底可言，而是在股票由弱走强的瞬间发出信号时做抄底买入。

在股票指标中有很多有关强弱的指标，如 KDJ，RSI，W&R 威廉指标等，这些指标的有效范围为 0~100。我们一般选择 KDJ 作为抄底指标，它的 J 值在数值 20 以下时是弱势股，在数值 80 以上时是强势股，正常波动在 20~80。因此，我们可以把 K 值上传 20 作为抄底买入信号。

KDJ 指标又叫随机指标，最早是以 KD 指标的形式出现的，而 KD 指标是在威廉指标的基础上发展起来的。不过，KD 指标只判断股票超买超卖的现象，而在 KDJ 指标中则融合了移动平均线速度上的观念，形成了比较准确的买卖信号依据。在实践中，K 线与 D 线配合 J 线组成 KDJ 指标来使用。在设计过程中，KDJ 指标主要是研究最高价、最低价和收盘价之间的关系，同时融合了动量观念、强弱指标和移动平均线的一些优点。因此，KDJ 指标能够比较迅速、快捷、直观地研判行情，并被广泛用于股市的中短期趋势分析中，是期货和股票市场上最常用的技术分析工具。

KDJ 指标一般是用于股票分析的统计体系，即根据统计学原理，通过一个特定的周期（常为 9 日、9 周等）内出现过的最高价、最低价和最后一个计算周期的收盘价，以及三者之间的比例关系，来计算最后一个计算周期的未成熟随机值“RSV”，

然后根据平滑移动平均线的方法来计算 K 值、D 值与 J 值，并绘成曲线图来研判股票走势。

在 KDJ 的计算中，首先要计算周期（n 日、n 周等）的 RSV 值，即未成熟随机指标值，再计算 K 值、D 值、J 值等。以 n 日 KDJ 数值的计算为例，其计算公式为

$$n \text{ 日 RSV} = (C_n - L_n) / (H_n - L_n) \times 100$$

公式中，C_n 为第 n 日收盘价；L_n 为 n 日内的最低价；H_n 为 n 日内的最高价。

其次，计算 K 值与 D 值：

$$\text{当日 K 值} = 2/3 \times \text{前一日 K 值} + 1/3 \times \text{当日 RSV}$$

$$\text{当日 D 值} = 2/3 \times \text{前一日 D 值} + 1/3 \times \text{当日 K 值}$$

若无前一日 K 值与 D 值，则可分别用 50 来代替。

$$J \text{ 值} = 3 \times \text{当日 K 值} - 2 \times \text{当日 D 值}$$

以 9 日为周期的 KD 线为例，即未成熟随机值，计算公式为

$$9 \text{ 日 RSV} = (C - L_9) \div (H_9 - L_9) \times 100$$

公式中，C 为第 9 日的收盘价；L₉ 为 9 日内的最低价；H₉ 为 9 日内的最高价。

$$K \text{ 值} = 2/3 \times \text{第 8 日 K 值} + 1/3 \times \text{第 9 日 RSV}$$

$$D \text{ 值} = 2/3 \times \text{第 8 日 D 值} + 1/3 \times \text{第 9 日 K 值}$$

$$J \text{ 值} = 3 \times \text{第 9 日 K 值} - 2 \times \text{第 9 日 D 值}$$

若无前一日 K 值与 D 值，则可以分别用 50 代替。

KDJ 指标的使用方法：

(1) K 与 D 的值永远介于 0 到 100 之间。当 D 大于 80 时，行情呈现超买现象。当 D 小于 20 时，行情呈现超卖现象。

(2) 在上涨趋势中，K 值大于 D 值，当 K 线向上突破 D 线时为买进信号。在下跌趋势中，K 值小于 D 值，当 K 线向下跌破 D 线时为卖出信号。

(3) K 与 D 的指标值不仅能反映出市场的超买、超卖程度，还能通过交叉突破发出买卖信号。

(4) K 与 D 的指标值不适于发行量小、交易不活跃的股票，但是它们对大盘和热门的大盘股有极高的准确性。

(5) 当随机指标与股价出现背离时，一般为转势的信号。

(6) K 值和 D 值上升或者下跌的速度减弱，倾斜度趋于平缓是短期转势的预警

信号。

由于 KDJ(9,9,3)默认参数太活跃, 因此我们调整 KDJ 参数来作为抄底指标。

根据我们的经验, 最好使用 KDJ(22,11,22)作为抄底指标, 因为仅仅用 KDJ 指标难免会出现错误抄底信号。上节讲过成交量的高低对其他指标分析有确认的作用, 因此我们对 KDJ 指标发出的抄底信号再做一次度量, 即满足成交量要求就保留抄底信号, 不满足成交量要求就过滤掉。

根据经验, 一般成交量一定要大于 91 日成交量均线, 股价才能够上涨。

抄底代码见示例 10-4。

```
#通过 KDJ 指标抄底
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pickle
import tushare as ts
import HP_global as g                                #全局变量命名空间
import HP_set                                          #初始化系统全局变量
from HP_formula import *                             #引入公式基本函数
from HP_draw import *                                #引入绘图函数

def KDJ(N=9, M1=3, M2=3):
    """
    KDJ 随机指标
    """
    RSV = (CLOSE - LLV(LOW, N)) / (HHV(HIGH, N) - LLV(LOW, N)) * 100
    K = EMA(RSV, (M1 * 2 - 1))
    D = EMA(K, (M2 * 2 - 1))
    J = K * 3 - D * 2
    return K, D, J

#我们设定股票池
zxcg=['601988','601328','601818','600015','601166','600909','601700']
##回测程序开始
date_s='2017-01-01'                                #回测开始时间
date_e='2019-01-04'                                #回测结束时间
plate=zxcg                                           #股票池
Acode=[]                                             #保存股票代码
Asdf = []                                           #保存股票数据
i=0
```

```

lmax=0
lendata=0
for code in plate:
    print(i,code)
    data = ts.get_k_data(code,ktype='D',start=date_s,end=date_e)
    lendata=len(data)
    if lendata<200:                                #数据少于 200 天
        continue
    if lmax<lendata:
        lmax=lendata
    i+=1
    mydf=data.copy()
    CLOSE=mydf['close']
    LOW=mydf['low']
    HIGH=mydf['high']
    OPEN=mydf['open']
    VOL=mydf['volume']
    C=mydf['close']
    L=mydf['low']
    H=mydf['high']
    O=mydf['open']
    V=mydf['volume']
    K,D,J=KDJ(22,11,22)
    mydf = mydf.join(pd.Series( K,name='K'))
    mydf = mydf.join(pd.Series( D,name='D'))
    mydf = mydf.join(pd.Series( J,name='J'))
    mydf['X20']=15
    mydf['X80']=80
    mydf['B1']=CROSS(J,mydf['X20'])  #J 上穿 20
    mydf['V91']=MA(VOL,91)
    mydf['TWO']=2
    mydf['ZERO']=0
    mydf['BB']=IF((CLOSE>OPEN)&(mydf['B1']>0) &(VOL>=mydf['V91']* 1.2),
mydf['TWO'],mydf['ZERO'])*50
    Acode.append(code)
    Asdf.append(mydf)
def DrawKDJ(i):                                #绘图函数
    Asdf[i]=Asdf[i].tail(50)
    plt.Figure()

```

```

plt.title(Acode[i])
ax1 = plt.subplot(211)
ax_K(ax1,Asdf[i],Acode[i],2)
B= list(Asdf[i].BB)
df_stockload=Asdf[i]
##循环方式实现
for j in range(len(B)):
    if B[j] > 0:
        ax1.annotate(u"金叉", xy=(j+352, df_stockload['high']
[j+352]*0.95), xytext=(j+352, df_stockload['high'][j+352]*0.7),
                    arrowprops=dict(facecolor='blue', shrink=0.1))
ax2 = plt.subplot(212)
Asdf[i].K.plot.line(legend=True)
Asdf[i].D.plot.line(legend=True)
Asdf[i].J.plot.line(legend=True)
Asdf[i].X80.plot.line(legend=True)
Asdf[i].X20.plot.line(legend=True)
Asdf[i].BB.plot.line(legend=True)
plt.show()
DrawKDJ(6)                                     #显示列表 6 数据

```

程序运行结果如图 10-4 所示，BB 指标大于 0 为买点信号。

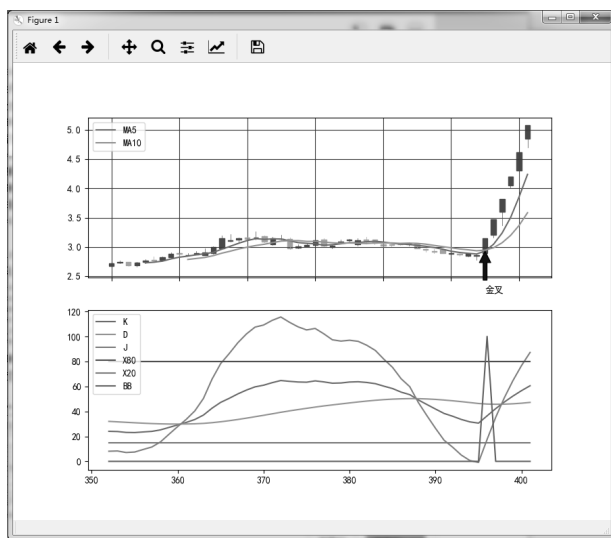


图 10-4 程序运行结果

10.4 持仓分析——卖点信号

持仓分析主要是分析持仓股票的状态，以决定何时卖出。

持仓股票卖出有指标提示卖出、止损卖出和止盈卖出。

指标提示卖出信号与指标买入信号的设计过程相似。一般我们采用什么方法买股，就用什么方法卖股。即如果我们用收盘价 5 日均线上穿 20 日均线作为买点，就把收盘价 5 日均线下穿 20 日均线作为卖点，如果我们用 KDJ 指标 J 值上穿 20 作为买点，就把 J 值下穿 80 作为卖点。

对于卖点信号，一般我们不用考虑成交量的变化，因为无论是缩量下跌还是放量下跌都要卖出股票。

有些投资者可能会问，万一卖掉股票又涨了怎么办？涨了就涨了，如果不涨，股价下跌得更快。天涯何处无芳草，何必单恋一枝花。绝对不能跟股票搞“单相思”，做股票要赚能看到的低风险收益，否则就犯了贪心的忌讳。

1. 股票卖出信号

卖出信号主要是通过分析所持股票判断是不是已经出现卖出信号了。如果出现信号，就及时卖出股票。

我们继续以 KDJ 指标作为卖出信号，使用 KDJ(22,11,22)指标的 J 值下穿数值 80 作为卖出信号。

卖出代码见示例 10-5。

```
#通过 KDJ 指标卖出
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pickle
import tushare as ts
import HP_global as g                                #全局变量命名空间
import HP_set                                          #初始化系统全局变量
from HP_formula import *                             #引入公式基本函数
from HP_draw import *                                #引入绘图函数
```

```

def KDJ(N=9, M1=3, M2=3):
    """
    KDJ 随机指标
    """
    RSV = (CLOSE - LLV(LOW, N)) / (HHV(HIGH, N) - LLV(LOW, N)) * 100
    K = EMA(RSV, (M1 * 2 - 1))
    D = EMA(K, (M2 * 2 - 1))
    J = K * 3 - D * 2
    return K, D, J

#我们设定股票池
zxg=['601988','601328','601818','600015','601166','600909','601700']
##回测程序开始
date_s='2017-01-01'                #回测开始时间
date_e='2019-01-04'                #回测结束时间
block=zxg                           #股票池
Acode=[]                            #保存股票代码
Asdf = []                           #保存股票数据
Asprice=[]                          #买入价格
Asnumber=[]                         #持股数量
datei=0                             #交易周期数
maxi=len(block)                     #股票池总数
i=0
lmax=0
lendata=0
for code in block:
    print(i,code)
    data = ts.get_k_data(code,ktype='D',start=date_s,end=date_e)
    lendata=len(data)
    if lendata<200:                  #数据少于 200 天
        continue
    if lmax<lendata:
        lmax=lendata
    i+=1
    mydf=data.copy()
    CLOSE=mydf['close']
    LOW=mydf['low']
    HIGH=mydf['high']

```

```

OPEN=mydf['open']
VOL=mydf['volume']
C=mydf['close']
L=mydf['low']
H=mydf['high']
O=mydf['open']
V=mydf['volume']
K,D,J=KDJ(22,11,22)
mydf = mydf.join(pd.Series( K,name='K'))
mydf = mydf.join(pd.Series( D,name='D'))
mydf = mydf.join(pd.Series( J,name='J'))
mydf['X15']=15
mydf['X80']=80
mydf['B1']=CROSS(J,mydf['X15'])      #J 上穿 10
mydf['S1']=CROSS(mydf['X80'],J)      #J 下穿 80
mydf['V91']=MA(VOL,91)
mydf['TWO']=2
mydf['ZERO']=0
mydf['BB']=IF((CLOSE>OPEN) & (mydf['B1']>0) & (VOL>=mydf['V91']*1.2) ,
mydf['TWO'],mydf['ZERO'])*50
mydf['SS']=100-50*mydf['S1']
#(CLOSE>OPEN) 阳线
Acode.append(code)
Asdf.append(mydf)
def DrawKDJ(i):                      #绘图函数
    #Asdf[i]=Asdf[i].tail(50)
    plt.Figure()
    plt.title(Acode[i])
    ax1 = plt.subplot(211)
    ax_K(ax1,Asdf[i],Acode[i],2)
    ax2 = plt.subplot(212)
    x=Asdf[i][Asdf[i].index%(int(len(Asdf[i])/8))==0]
    date_tickers=x['date']          #刻度值
    Asdf[i].K.plot.line(legend=True)
    Asdf[i].D.plot.line(legend=True)
    Asdf[i].J.plot.line(legend=True)
    Asdf[i].X80.plot.line(legend=True)

```



```

Asdf[i].X15.plot.line(legend=True)
Asdf[i].BB.plot.line(legend=True)
Asdf[i].SS.plot.line(legend=True)
ax2.set_xticks(range(len(date_tickers)))
ax2.set_xticklabels(date_tickers)
ax2.grid(True, color='r')
ax2.xaxis.set_major_locator(mticker.MaxNLocator(8)) #x 轴分成几等份
plt.show()
DrawKDJ(6)                                     #显示列表 6 数据

```

程序运行结果如图 10-5 所示。

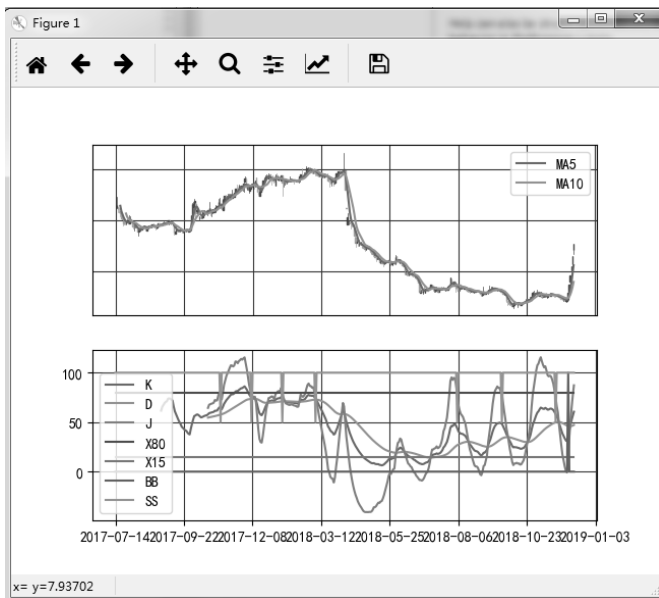


图 10-5 程序运行结果

从程序输出结果中可以看出，因为没有经过附加条件过滤，所以卖点信号比较多。读者可以根据自己的经验，增加一些卖出信号的过滤条件。

2. 股票止损信号

在股票软件公式平台上，我们做过数百种不同的股票分析指标。由于股票软件仅仅能实现选股、预警和预测买卖点信号，无法记录持仓和成本，因此股票软件就没有止损可言。而 Python 不但能够同时分析多只股票，而且还能够记录交易信息，

如成本价等，这就是量化分析系统的魅力。

技术分析的结果是准和不准的问题，也就是准确率的问题。而 100% 准确的技术分析只可能会在某个市场或市场的某个阶段存在。例如，牛市选股策略只适合牛市环境选股，如果用于熊市操作，就只能不断亏钱。尽管投资者也清楚牛市选股策略的规则，但他们也不好精确判断。人尚且如此，何况自动交易机器呢！

顾名思义，股票止损就是为了减少因股价继续下跌带来的亏损而进行的亏钱卖出的操作。

股票止损操作包括价格止损、基本面止损和指标止损。

1) 价格止损

价格止损有买进价格止损和成本价格止损。

买进价格止损一般以买进价格的一定下跌比例作为止损点，超过止损点就卖出。买进价格止损通常是对一只股票进行全买或全卖的操作，止损比例通常受股票振幅的影响，一般止损点的价格比买入的价格低 5%。

买进价格止损的计算公式：

买进价格止损价格 = 买进价格 * (1 - 止损比例)

例如，以 8 元/股买进某只股票，止损点为 5%，止损价格为 7.6 ($8 * (1 - 0.05)$) 元/股。即当股票价格跌破 7.6 元/股时，就卖掉全部该只股票。

成本价格止损是指对计划中长期持有某只股票分步买进，分步卖出。由于买进价格不同且夹杂着红股和红利等因素或者按照某种买卖策略分批买进某只股票造成股票成本价格波动，这时只能采用成本价格止损。

成本价格止损相关的计算公式：

成本价格止损价格 = ((买入花费金额 - 卖出所得金额 + 红利收入) / 股票余额数量) * (1 - 止损比例)

买入花费金额 = 买入股票价格 * 股票数量 + 交易成本

卖出所得金额 = 卖出股票价格 * 股票数量 - 交易成本

交易成本 = 佣金 + 印花税 + 经手费 + 规费 + 其他费用

交易成本与交易标的有关，即便是买卖 A 股，也与所签订的佣金协议有关。其中是净佣金，还是包含常规等费用的全佣金，它们的计算公式不同。

例如，A 股协议佣金为净佣金万三，卖出印花税（单向，卖出交）为 0.1%，经手费（双向）为成交金额的 0.00487%，证管费（双向）为成交金额的 0.002%，红利税为 20%。如果客户 1 个月内多次买卖过某只股票，如 2 日买入 10000 股，成交

价格为 10 元/股；8 日买入 5000 股，成交价格为 10.5 元/股；15 日卖出 7000 股，卖出价格为 11 元/股；18 日每股分红 1 元现金，20 日到帐；22 日买入 2000 股，成交价格为 10.5 元/股。假若成本价格止损比例为 5%，问止损价格为多少元？

交易情况表，见表 10-1。

表 10-1 交易情况表

| 日期 | 交易 | 交易金额/元 | 股票余额/股 |
|-----------|---|------------|--------|
| 2 日 | $10000 \times 10 \times (1 + 0.0003 + 0.0000487 + 0.00002) = 100036.87$ | -100036.87 | 10000 |
| 8 日 | $5000 \times 10.5 \times (1 + 0.0003 + 0.0000487 + 0.00002) = 52519.152$ | -52519.152 | 15000 |
| 15 日 | $7000 \times 11 \times (1 - 0.0003 - 0.0000487 - 0.00002 - 0.001) = 76687.38$ | 76687.38 | 8000 |
| 18 日-20 日 | $8000 \times 1 \times (1 - 0.2) = 6400$ | 6400.00 | 8000 |
| 22 日 | $2000 \times 10.5 \times (1 + 0.0003 + 0.0000487 + 0.00002) = 21007.728$ | -21007.728 | 10000 |
| 汇总 | (减号“-”代表支出；不带“-”表示收入) | -90476.36 | 10000 |

根据表 10-1，我们计算股票的成本价格。

持仓该股成本 = $100036.87 + 52519.152 - 76687.38 - 6400.00 + 21007.728$

$= 90476.36$ (元)

持仓该股成本价格 = 持仓该股成本 / 股票余额 = $90476.36 / 10000 = 9.05$ (元)

成本止损价格 = 持仓该股成本价格 * (1 - 止损比例) = $9.05 \times (1 - 0.05) = 8.60$ (元)

即止损价格为 8.06 元。

2) 基本面止损

基本面止损需要用户手动强制平仓。例如，所持有的某只股票，突然有了利空消息，或者业绩变脸带上了 ST 帽子，这时持股用户多数会不计成本进行平仓，因此这类股票就会出现连续跌停的情况。

我们在建立自动交易系统时，除了要建立备选股票池外，还要设置黑名单股票池。这样在软件自动选股时就会先剔除掉黑名单股票池中的股票代码，避免参与过大风险的股票。

在交易过程中，如果发现量化系统所持股票被设置为黑名单，系统就应该优先去卖掉所有黑名单中的股票。

3) 指标止损

指标止损也分为两种情况：一是对趋势指标自动具有止损、止赢的性质；二是对强弱类抄底指标没有止损性质，只能利用持续跟踪抄底条件是否满足作为止损条件。

例如，均线指标收盘价 5 日均线上穿 20 日均线买入股票，一段时间后收盘价 5 日均线下穿 20 日均线卖出。如果卖出价格高于买入价格，就可以看作止赢；如果卖出价格低于买入价格，就可以看作止损。

强弱指标如长周期 KDJ 指标，我们设定 J 值上穿 15 作为抄底买入条件，J 值下穿 85 作为卖出条件，J 值以 50 为界划分强弱区间。J 值能从 85 下穿，在多数情况下股票都会有盈利，且判断错误的股票 J 值上 50 都困难，更不可能上 85 了，所以不能把 J 值从 85 下穿作为止损条件，因此可以用抄底条件是否成立作为止损条件。我们把 J 值上穿 15 作为由弱转强的起点，但是不久 J 值又下穿 15，这说明我们在判断底部位置时出错了，即抄底条件不成立，因此要止损卖出股票。

10.5 操作策略

操作策略可以理解为投资策略。

为了确保盈利，要制定一个操作策略。操作策略包含分析技术和操作方法。

操作策略有很多，在不能保证每次操作都能有收益的前提下，目的都是大赚而小赔、赢多而亏少。

随着证券市场的发展和完善，股市信息逐步对称发展，也就是说在股市中信息传递到机构和散户的时间是相同的。很多中小投资者也承认这一点，但是他们仍然抱怨资金太少，与机构相比没有资金上的优势。我们都听过“田忌赛马”的故事，这个故事生动地告诉我们：在实际的工作生活中，我们要根据规则科学地调整内部次序和策略，从而在整体上发挥最大的功效。

股市中的操作也是如此。在一定的前提下，投资者也可以利用好的操作策略在股市中经常获利。

股票的操作策略有很多，下面介绍几种常见的。

1. 投资三分法

投资三分法是投资者将金融资产分配在不同形态上的一种方法，也是进行证券投资的一种策略。这种方法在西方国家较为流行。

投资三分法的具体操作：将全部资产的 1/3 存入银行以备不时之需，1/3 用来购买债券、股票等有偿证券作为长期资本，1/3 用来购置房产、土地等不动产。在上述

资产分布中,存入银行的资产具有较高的安全性和变现力,但缺乏收益性;投入有价证券的金融资产虽然有效好的收益,但却具有较高的风险;投资于房地产的资金一般也会增值,但又缺乏变现力。如果将全部资产合理地分布在上述三种形态上,则可以相互补充、相得益彰。

在有价证券的投资上,也可以实行三分法。一部分购买债券或优先股票,一部分投资普通股,另一部分作为预备金或准备金,以备机动运用。在这种三分法中,虽然投资债券的部分获利不大,但比较安全可靠。因此,一般投资者都愿意拥有这部分安全可靠的债券。虽然购买股票风险较高,但往往能够获得比较优厚的红利收入,甚至还能获得较为可观的买卖差价收入。因此,它也颇受投资者欢迎。而保留一部分资金作为准备金,则可以在股票市场上出现较好的投资机会时进行追加投资,也可以在投资失利时用作失利后的补充和承担损失的能力准备。

证券投资三分法兼顾了证券投资的安全性、收益性和流动性的三原则,是一种颇具参考性的投资组合与投资技巧。

平衡型股票基金的投资策略就采用了这类方法。

2. 排列组合法

排列组合法是投资者运用科学的方法将股票基本面与价位进行全方位的排列组合,并据此进行股票买卖的方法。股票基本面与价位的排列组合,一般有四种情形:

- (1) 基本面好,价位高。
- (2) 基本面好,价位低。
- (3) 基本面差,价位高。
- (4) 基本面差,价位低。

这里所讲的股票基本面差,除了包含公司的收益或股利、公司的营运能力、获利能力、未来展望等之外,还包含是否所属当前的热点板块或炒作概念。通过这四种排列组合的架构,即可将所有上市的股票予以归类。

一般来说,第一类基本面好、价位高和第四类基本面差、价位低的股票,应该算是名副其实。因此,这两类股票不可能出现大幅度的波动。至于第二类基本面好、价位低和第三类基本面差、价位高的股票,由于名不副实,因此将会出现调整的可能。因为基本面好的股票,其价位都落后于其他,投资报酬率势必显得突出,游资就会自然而然地往这些投资报酬高的地方流,所以此类股票价格极易获得补涨。至于基本面差、价位高的股票,也必将要调整到基本面与价值相一致的水平。

将股票的基本面与价位进行排列组合后，其买卖策略是适时卖出第三类的股票，并买进第二类的股票。

股票的这种投资组合策略，比较适合量化投资组合。

3. 等级投资计划法

等级投资计划法又称尺度法，是进行股票投资的方法之一。其具体操作是在确定以某种股票作为买卖对象之后，继而确定所选股票市场变动的某一等级作为买卖时机，之后当股价下降一个等级时，就买进预先确定的一个单位的股数；当股价上升一个等级时，就出售一个单位的股数。

例如，某投资者选定 A 公司的股票作为投资对象，确定股价每变动一单位为一个等级，每次买卖 1000 股。若第一次按每股 10 元买进 1000 股，在股价降到 9 元时，再买进 1000 股；在降到 8 元时，又买进 1000 股。假设现在市场价格开始上升，当价格上升到每股 9 元时卖出 1000 股，当价格上升到 10 元时再卖出 1000 股，当上升到 11 元时，再卖出 1000 股。其买卖结果如表 10-2 所示。

表 10-2 等级投资计划法的交易情况

| 股价/元 | 买进股数/股 | 买进金额/元 | 卖出股数/股 | 卖出金额/元 | 盈亏/元 |
|------|--------|---------|--------|--------|-------------------------------|
| 10 | 1000 | 10000 | | | 0 |
| 9 | 1000 | 9000 | | | $2000*9-(10000+9000)=-1000$ |
| 8 | 1000 | 8000 | | | $3000*8-27000=-3000$ |
| 9 | | | 1000 | 9000 | $2000*9-27000+1000*9=0$ |
| 10 | | | 1000 | 10000 | $1000*10-18000+1000*10=2000$ |
| 11 | | | 1000 | 11000 | $1000*11-(1000*10-2000)=3000$ |
| 合计 | 3000 | 27000 元 | 3000 | 30000 | 操作这个股票总计盈利 3000 元 |

表 10-2 的最终交易结果表明，共计投入资金 27000 元，卖出后收回资金 30000 元，收支相抵后尚有盈余 3000 元（未除去手续费等成本开支）。买进后，尽管股票持续下跌，但是通过这种方法最终还是会获利。

需要指出的是，等级投资计划法不宜在股价持续上涨或持续下跌的行情中运用。如果股价在较长时间内是持续上涨的，那么这种分段抛售的办法就会使投资者失去本来可以获取的高利润；反之，如果股价在较长时间内是持续下跌的，投资者要是按照事先确定的分级标准不断地购买，就可能在高价被“套牢”，而失去股票出手的机会。

因此，等级投资计划法只适合在震荡幅度较大的市场中操作，不适合在趋势确定的牛市或熊市中操作。客户基金定投就采用了这种方法。

4. 金额平均法

金额平均法又称均价成本投资法、平均资金投资计划法和固定投入法。它是在一定时期内，固定一定量的资金分期平均购买某种或一些股票的投资方法，这种方法适合量化投资操作。

金额平均法原来只用于一只股票的等金额买入，其具体的操作是选定某种具有长期投资价值且价格波动较大的股票，在一定的投资时间内，无论股值是上涨还是下跌，都坚持定期以相同的资金购入该种股票。

例如，某投资者每月投入 1000 元（或大约 1000 元）用于购买某只股票，示例演示允许买入零股（A 股交易规则，最低买 1 手，100 股），5 个月后所购买的股票情况如表 10-3 所示。

表 10-3 金额平均法的交易情况

| 购买时间 | 市价/元 | 购入股数/股 | 累计购入 | 总投资数额/元 | 股票市值/元 |
|------|------|--------|------|---------|--------|
| 1 月 | 40 | 25 | 25 | 1000 | 1000 |
| 2 月 | 50 | 20 | 45 | 2000 | 2250 |
| 3 月 | 30 | 33 | 78 | 2990 | 2340 |
| 4 月 | 50 | 20 | 98 | 3990 | 4900 |
| 5 月 | 60 | 16 | 114 | 4950 | 6480 |

表 10-3 表明，由于各个时期的股价不同，因此所购买的股票的数量也不相同。

5 个月后的每股平均股价为 $46[(40+50+30+50+60)/5=46]$ （元）。

而投资者每股的平均投资成本为 $43.42(4950/114=43.42)$ （元）。

这样，投资者累计购入的每股价格成本值就低于每股的简单平均市场股价。造成这种情况的原因在于，每一时期的投资总额为一定数额，而当股票的市价越低时，所购买的股票数量就越多，其结果必然是在总持有股数中，低价所购的股票数量所占的比例越大，而高价所购的股票数量所占的比例越小，因此其平均购买值就会低于平均市场价。

金额平均法的优点：

- （1）方法简单，投资者只需要定期定额投资，不必考虑投资时间的确定问题。
- （2）既可以避免在高价时买进过多股票的风险，又可以在股票下跌时有机会购

进更多的股票。

(3) 少量资金便可进行连续投入，并可享受股票长期增值的利益。

在采用这种方法时应注意三点：一是选择经营稳定、利润稳定上升的公司的优良股票。二是有一个较长的投资时间。如果期限较短，效果就不会很明显。三是对于价格波动幅度较大且股价呈上升趋势的股票，很可能会发生投资亏损。

随着量化投资的兴起，金额平均法已逐渐成为量化投资中资金自动分配的一种主要方法。即量化系统分配数个交易单元，每个交易单元平分剩余资金，将这个交易单元所分配的资金买入选定的一只股票，第 2 个交易单元再用同样方法买入另外一只股票，其他交易单元以此类推。

当所有单元都买入股票后，就不再操作，一直等某个单元的股票卖出后，再去找新的股票买入。

多只股票回测单元就是利用的金额平均法策略。

10.6 多只股票量化回测

这节主要讲解 Python 的多只股票量化回测程序。

我们设计一个策略模型进行回测，买卖点指标用长周期 KDJ，止损按买入价格下跌幅度的 5%。交易模型的原理如下。

(1) 选择沪深 300 指数的成分股作为交易目标股票。

(2) 把总资金划分为 20 个交易单元。每次交易使用的资金=剩余资金/剩余交易单元数。

(3) 买入：选股采用简单的轮盘方式，股票代码首尾相接。即一个交易单元当日没有买点，就按沪深 300 指数股票的顺序选择下一只股票。检测是出现买点信号，这样每只股票都有机会被买入。如果持仓单元满了，就买入轮盘，转动结束。如果查询结束没有出现买点的股票，就等下一个交易日继续寻找出现买点的股票。

(4) 止损：每天检查持仓单元是否发出止损信号。如果出现信号，就卖出全部股票，将资金释放回总资金池。

(5) 卖出：每天检查持仓单元是否发出指标卖出信号。如果出现信号，就卖出全部股票，将资金释放回总资金池。

(6) 卖出股票后，剩余持仓单元数加 1，腾出的持仓单元继续寻找股票买入。

(7) 买入资金=总资金池余额/剩余持仓单元数。

(8) 进入下一个交易日操作，重复上述操作直到最后交易日。

下面见回测示例 10-6。

```
#通过 KDJ 指标卖出
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pickle
import tushare as ts
import HP_global as g                                #全局变量命名空间
import HP_set                                          #初始化系统全局变量
from HP_formula import *                             #引入公式基本函数
from HP_draw import *                                #引入绘图函数
def KDJ(N=9, M1=3, M2=3):
    """
    KDJ 随机指标
    """
    RSV = (CLOSE - LLV(LOW, N)) / (HHV(HIGH, N) - LLV(LOW, N)) * 100
    K = EMA(RSV, (M1 * 2 - 1))
    D = EMA(K, (M2 * 2 - 1))
    J = K * 3 - D * 2
    return K, D, J
hs300=ts.get_hs300s()                                #沪深 300
##回测程序开始
date_s='2016-01-01'                                #回测开始时间
date_e='2016-12-31'                                #回测结束时间
block=list(hs300.code)                              #股票池
datei=0                                              #交易周期数
maxi=300                                            #股票池总数
zzj=100000000.00                                  #初始总资金
gps=20                                              #最大持仓股票数
stamp_duty=0.001                                   #印花税 0.1%
trading_Commission=0.0005                          #交易佣金 0.05%
stop_loss_range=0.05                               #止损幅度
#-----
```

```

jy=""                                ##交易记录
jycs=0
ars=[]                               #保存股票代码
art=[]                               #保存持仓状态
arr = []                             #保存股票数据
i=0
lmax=0
lendata=0
for code in block:
    print(i,code)
    data = ts.get_k_data(code,ktype='D',start=date_s,end=date_e)
    lendata=len(data)
    if lendata<200:                   #数据少于 200 天
        continue
    if lmax<lendata:
        lmax=lendata
        datem=data.date
    i+=1
    mydf=data.copy()
    CLOSE=mydf['close']
    LOW=mydf['low']
    HIGH=mydf['high']
    OPEN=mydf['open']
    VOL=mydf['volume']
    C=mydf['close']
    L=mydf['low']
    H=mydf['high']
    O=mydf['open']
    V=mydf['volume']
    K,D,J=KDJ(22,11,22)
    mydf = mydf.join(pd.Series( K,name='K'))
    mydf = mydf.join(pd.Series( D,name='D'))
    mydf = mydf.join(pd.Series( J,name='J'))
    mydf['X15']=15
    mydf['X85']=85
    mydf['B1']=CROSS(J,mydf['X15'])  #J 上穿 10
    mydf['S1']=CROSS(mydf['X85'],J)  #J 下穿 80

```

```

mydf['V91']=MA(VOL,91)
mydf['TWO']=2
mydf['ZERO']=0
mydf['BB']=IF((CLOSE>OPEN)&(mydf['B1']>0)&(VOL>=mydf['V91']*1.2),
mydf['TWO'],mydf['ZERO'])*50
mydf['SS']=mydf['S1']
#(CLOSE>OPEN) 阳线
arr.append(mydf)
ars.append(code)
art.append(0)
i=i+1
if (i>maxi):                                #超过最大股票数量就退出
    break
#-----
sts=[]                                       #持股数量
stp=[]                                       #持股价格
sti=[]                                       #持股编号
stn=[]                                       #股票代码
stpn=[]                                     #股票最新价格
dates=len(datem)
datei=0                                     #交易周期数
maxi=i                                     #股票池总数
i=0
while i<gps+2:
    sts.append(0)
    sti.append(0)
    stn.append('')
    stp.append(0.00)
    stpn.append(0.00)
    i=i+1
gpn=0                                       #目前股票持仓数量
zjm=zzj                                     #总资金
xi=1                                       #当前股票位置
#-----
hl=[zzj]
#-----
jy=jy+'-----开始回测-----\n'

```

```

print('-----开始回测-----')
while datei<dates :                                #还在交易周期中
    dated=datem[datei]
    #卖出
    j=1                                              #持股指针
    while j<=gpn:
        ii=sti[j]                                  ##股票数据索引号
        try:
            jj=arr[ii][arr[ii].date==dated].index[0]
        except :
            jj=-1
        if jj>=0 :                                  #查找是否存在开盘数据
            stpn[j]=arr[ii].close[jj]
            if stpn[j]<(stp[j]*(1-stop_loss_range)):
                ss='止损'
                sp=arr[ii].close[jj]    #卖出价格
                zjm=zjm+sp*sts[j]*(1.00-trading_Commission-stamp_duty)
                print(dated+' 单元['+str(j)+'] '+ss+' '+ars[ii]+' 数量:
'+str(sts[j])+ ' 价格:'+str(round(sp,2))+ ' 盈亏:'+str(round((sp-stp[j])/stp[j],2))+ '
资金余额:'+str(round(zjm,2)))
                jy=jy+dated+' 单元['+str(j)+'] '+ss+' '+ars[ii]+' 数量:
'+str(sts[j])+ ' 价格:'+str(round(sp,2))+ ' 盈亏:'+str(round((sp-stp[j])/stp[j],2))+ '
资金余额:'+str(round(zjm,2))+ '\n'
                jycs=jycs+1
                art[ii]=0
            if j<gpn:
                sti[j]=sti[gpn]
                sts[j]=sts[gpn]
                stp[j]=stp[gpn]
                stn[j]=stn[gpn]
                stpn[j]=stpn[gpn]
                gpn=gpn-1
                j=j-1
            else:
                gpn=gpn-1
                j=0
        elif arr[ii].SS[jj]>0 :    #发出卖点

```

```

        ss='卖出'
        sp=arr[ii].close[jj]    #卖出价格
        zjm=zjm+sp*sts[j]*(1.00-trading_Commission-stamp_duty)
        print(dated+' 单元['+str(j)+'] '+ss+' '+ars[ii]+' 数量:
'+str(sts[j])+ ' 价格:'+str(round(sp,2))+ ' 盈亏:'+str(round((sp-stp[j])/stp[j],2))+
'资金余额:'+str(round(zjm,2)))
        jy=jy+dated+' 单元['+str(j)+'] '+ss+' '+ars[ii]+' 数量:
'+str(sts[j])+ ' 价格:'+str(round(sp,2))+ ' 盈亏:'+str(round((sp-stp[j])/stp[j],2))+
'资金余额:'+str(round(zjm,2))+'\n'
        jycs=jycs+1
        art[ii]=0
        if j<gpn:
            sti[j]=sti[gpn]
            sts[j]=sts[gpn]
            stp[j]=stp[gpn]
            stn[j]=stn[gpn]
            stpn[j]=stpn[gpn]
            gpn=gpn-1
            j=j-1
        else:
            gpn=gpn-1
            j=0
        j=j+1
    #买入
    gpi=gps-gpn                    #空置单元数
    if gpi>0 :
        mrzj=zjm/gpi                #每个单元能使用的资金
    else:
        mrzj=zjm
    ii=xi
    j=gpn
    while j<gps :
        try:
            jj=arr[ii][arr[ii].date==dated].index[0]
        except :
            jj=-1
        if jj>=0 :

```

```

        if arr[ii].BB[jj]>0 :          #发出买点
            ss='买入'
            bp=arr[ii].close[jj]      #买入价格
            x=int(mrzj/(bp*(1+trading_Commission)))/100
            gpn=gpn+1
            sts[gpn]=x*100.00
            sti[gpn]=ii
            stp[gpn]=bp
            stn[gpn]=ars[ii]
            stpn[gpn]=bp
            art[j]=1
            zjm=zjm-x*100*bp*(1.00+trading_Commission)
            j=j+1
            jy=jy+dated+' 单元['+str(gpn)+'] '+ss+' '+stn[gpn]+' 数量:
'+str(sts[gpn])+ ' 价格:'+str(bp)+' 资金余额:'+str(round(zjm,2))+'\n'
            print(dated+' 单元['+str(gpn)+'] '+ss+' '+stn[gpn]+' 数量:
'+str(sts[gpn])+ ' 价格:'+str(bp)+' 资金余额:'+str(round(zjm,2)))
            jycs=jycs+1

        ii=ii+1
        if gpn>=gps:
            break
        if ii>maxi:
            ii=1
        if ii==xi:
            break

    z=zjm
    k=0
    while k<gpn:
        z=z+sts[k]*stp[k]
        k=k+1
    hl.append(z)
    xi=ii
    datei=datei+1
    jy=jy+'-----回测结束-----\n'
    jy=jy+'资金余额: '+str(zjm)+'\n'
    jy=jy+'-----持股 '+str(gpn)+'只-----\n'
    print('-----回测结束-----')
```

```

print('资金余额: '+str(round(zjm,2)))
print('-----持股 '+str(gpn)+'只-----')
zj=zjm
i=1
while i<=gpn:
    print(' 单元['+str(i)+'] '+' 股票 '+stn[i]+' 数量 '+str(sts[i])+ ' 买
入价格:'+str(stp[i])+ ' 最新价格:'+str(stpn[i]) + ' 盈亏:'+str(round((stpn[i]-
stp[i])/stp[i],2)))
    jy=jy+' 单元['+str(i)+'] '+' 股票 '+stn[i]+' 数量 '+str(sts[i])+ ' 买
入价格:'+str(stp[i])+ ' 最新价格:'+str(stpn[i]) + ' 盈亏:'+str(round((stpn[i]-
stp[i])/stp[i],2))+'\n'
    zj=zj+sts[i]*stpn[i]
    i=i+1
hll=(zj-zzj)/zzj*100.00
print('-----回测结果-----')
jy=jy+'-----回测结果-----\n'
print(date_s+' 初始资金:'+str(zzj)+' 交易单元:'+str(gps))
jy=jy+date_s+' 初始资金:'+str(zzj)+' 交易单元:'+str(gps)+'\n'
print(date_e+' 总市值: '+str(zj)+' 获利率:'+str(round(hll,2))+ '% 交易
次数:'+str(jycs)+'\n')
jy=jy+date_e+' 总市值: '+str(round(zj,2))+' 获利率:'+str(round(hll,2))+ '%
交易次数:'+str(jycs)+'\n'

```

程序运行结果较长, 以下为最后部分的显示结果。

```

...
2016-12-08 单元[4] 卖出 601985 数量:76200.0 价格:6.84 盈亏:0.04 资金余额:
5903080.21
2016-12-08 单元[8] 卖出 600886 数量:81100.0 价格:6.52 盈亏:0.03 资金余额:
6430735.14
2016-12-12 单元[3] 止损 600705 数量:73900.0 价格:6.25 盈亏:-0.07 资金余额:
6891917.33
2016-12-14 单元[4] 止损 600066 数量:27400.0 价格:18.07 盈亏:-0.08 资金余额:
7386402.09
2016-12-15 单元[5] 卖出 601628 数量:25200.0 价格:23.51 盈亏:0.13 资金余额:
7977864.76
2016-12-16 单元[2] 卖出 601633 数量:50800.0 价格:10.53 盈亏:-0.0 资金余额:
8511986.37
2016-12-21 单元[5] 买入 600309 数量:31800.0 价格:16.716 资金余额:

```

```

7980151.79
    2016-12-21  单元[6] 买入 600221  数量:162700.0  价格:3.268  资金余额:
7448182.34
    2016-12-27  单元[7] 买入 600170  数量:139700.0  价格:3.806  资金余额:
6916218.29
-----回测结束-----
资金余额: 6916218.29
-----持股 7 只-----
单元[1]  股票 600050 数量 112600.0 买入价格:4.342 最新价格:7.28 盈亏:0.68
单元[2]  股票 600297 数量 74600.0 买入价格:6.465 最新价格:6.311 盈亏:-0.02
单元[3]  股票 600690 数量 53000.0 买入价格:9.864 最新价格:9.545 盈亏:-0.03
单元[4]  股票 600309 数量 30900.0 买入价格:15.961 最新价格:16.928 盈
亏:0.06
单元[5]  股票 600309 数量 31800.0 买入价格:16.716 最新价格:16.928 盈
亏:0.01
单元[6]  股票 600221 数量 162700.0 买入价格:3.268 最新价格:3.19 盈亏:-0.02
单元[7]  股票 600170 数量 139700.0 买入价格:3.806 最新价格:3.712 盈
亏:-0.02
-----回测结果-----
2016-01-01  初始资金:10000000.0  交易单元:20
2016-12-31  总市值: 10811596.888950005  获利率:8.12%  交易次数:131
    
```

从上面的运行结果中我们可以看出，有买入、卖出和止损的操作，程序完全自动交易。

10.7 深度学习预测股价

传统股价的高度和转折点的预测是基于经验算法公式，或者利用黄金分割线等进行的。近年来，随着深度学习的兴起，出现了很多神经网络模型，它们也逐渐被用于金融投资分析上。

深度学习从开始到建模的一般流程：获取数据—数据预处理—训练建模—模型评估—预测，分类。

Python 常用的深度学习包有 Keras 包和 TensorFlow 包。

深度学习神经网络涉及的知识较多，下面为 LSTM 神经网络预测股票价格的例子。

见示例 10-7。


```

import matplotlib.pyplot as plt
import numpy as np
import math
import time
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import LSTM
import tushare as ts
'''

```

LSTM 神经网络

Long Short Term Memory network (LSTM) 是一种特殊的 RNNs, 可以很好地解决长时依赖问题。

```

'''
def new_dataset(dataset, step_size):
    data_X, data_Y = [], []
    for i in range(len(dataset)-step_size-1):
        a = dataset[i:(i+step_size), 0]
        data_X.append(a)
        data_Y.append(dataset[i + step_size, 0])
    return np.array(data_X), np.array(data_Y)
np.random.seed(7)
ds='2018-01-01' #开始日期
de=time.strftime('%Y-%m-%d',time.localtime(time.time())) #今天的日期
stockname='600059'
dataset=ts.get_k_data(stockname,ktype='D',start=ds,end=de)
print(dataset)
dataset=dataset[['open','high', 'low', 'close']]
dataset = dataset.reindex(index = dataset.index[::-1])
obs = np.arange(1, len(dataset) + 1, 1)
OHLC_avg = dataset.mean(axis = 1)
HLC_avg = dataset[['high', 'low', 'close']].mean(axis = 1)
close_val = dataset[['close']]
plt.suptitle(stockname,color='blue')
plt.plot(obs, OHLC_avg, 'r', label = 'OHLC avg')
plt.plot(obs, HLC_avg, 'b', label = 'HLC avg')
plt.plot(obs, close_val, 'g', label = 'Closing price')
plt.legend(loc = 'upper right')
plt.show()

```

```

OHLC_avg = np.reshape(OHLC_avg.values, (len(OHLC_avg),1)) #1664
scaler = MinMaxScaler(feature_range=(0, 1))
OHLC_avg = scaler.fit_transform(OHLC_avg)
train_OHLC = int(len(OHLC_avg) * 0.75)
test_OHLC = len(OHLC_avg) - train_OHLC
train_OHLC, test_OHLC = OHLC_avg[0:train_OHLC,:], OHLC_avg[train_OHLC:
len(OHLC_avg),:]
trainX, trainY = new_dataset(train_OHLC, 1)
testX, testY = new_dataset(test_OHLC, 1)
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
step_size = 1
model = Sequential() #定义
model.add(LSTM(32, input_shape=(1, step_size), return_sequences = True))
#添加 LSTM 神经网络, 即在头文件中已经导入了 LSTM, 输入神经网络数据形状为 (1, step_size)
#其中 32 为神经元个数
model.add(LSTM(16))
model.add(Dense(1)) #定义输出层的神经元个数为 1 个, 即输出只有一维
model.add(Activation('linear')) #根据情况添加激活函数
#模型编译和培训
model.compile(loss='mean_squared_error', optimizer='adagrad') #Try SGD,
adam, adagrad and compare!!!
model.fit(trainX, trainY, epochs=5, batch_size=1, verbose=2)
#最后一句是对模型进行传值 train_X, train_y, 定义 epochs 迭代次数等
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train RMSE: %.2f' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test RMSE: %.2f' % (testScore))
#创建类似的数据集训练预测
trainPredictPlot = np.empty_like(OHLC_avg)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[step_size:len(trainPredict)+step_size, :] = trainPredict
#创建类似 DATASSET 的测试预测
testPredictPlot = np.empty_like(OHLC_avg)

```

```

testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(step_size*2)+1:len(OHLC_avg)-1, :]
= testPredict
#反规范化主要数据集
OHLC_avg = scaler.inverse_transform(OHLC_avg)
print(testPredictPlot)
plt.plot(OHLC_avg, 'g', label = 'original dataset')
plt.plot(trainPredictPlot, 'r', label = 'training set')
plt.plot(testPredictPlot, 'b', label = 'predicted stock price/test set')
plt.legend(loc = 'upper right')
plt.xlabel('Time in Days')
plt.ylabel('OHLC Value of Apple Stocks')
plt.show()
#预测未来值
last_val = testPredict[-1]
last_val_scaled = last_val/last_val
next_val = model.predict(np.reshape(last_val_scaled, (1,1,1)))
print( "Last Day Value:", np.asscalar(last_val))
print( "Next Day Value:", np.asscalar(last_val*next_val))

```

程序运行结果，见图 10-6。



图 10-6 程序运行结果

```
Last Day Value: 9.474600791931152
```

```
Next Day Value: 10.230703353881836
```

10.8 股票数据网络爬虫

我们前面使用了金融 API 接口，但有些财经数据没有现成的 API 接口，那就可以考虑用 Python 做网络爬虫程序，即从网页中抓取所需的数据，但网站数据采集要遵守相关法律的规定。

我们以抓取股票行情数据为例。

网站选择的原则：股票信息静态存在于 html 页面中，非 JS 代码生成，没有 Robbts 协议限制。

选取方法：打开网页，查看源代码，搜索网页的股票价格数据是否存在于源代码中。

Beautiful Soup 是 Python 的一个库，最主要的功能是从网页中抓取数据。Beautiful Soup 提供一些简单的 Python 式的函数来处理导航、搜索、修改分析树等。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，由于比较简单所以不需要多少代码就可以写出一个完整的应用程序。

Beautiful Soup 自动将输入文档转换为 Unicode 编码，输出文档转换为 utf-8 编码。你不需要考虑编码方式，除非文档没有指定编码方式。如果没有指定编码方式，Beautiful Soup 就不能自动识别，然后通过说明一下原始编码方式即可解决这个问题。

Beautiful Soup 的安装：

```
pip install beautifulsoup4
```

导入 Beautiful Soup 库，库名为 bs4，导入格式如下：

```
from bs4 import BeautifulSoup
```

创建 beautifulsoup 对象：

```
soup = BeautifulSoup(html)
```

Beautiful Soup 将复杂的 HTML 文档转换成一个复杂的树形结构，每个节点都是

Python 对象。所有对象可以归纳为以下四种。

Tag: HTML 中的一个标签。

NavigableString: 标签内部的文字。

BeautifulSoup: 表示的文档内容。

Comment: 一个特殊类型的 NavigableString 对象，输出不包括注释符号的内容。

下面是一个股票代码网页抓取的例子：一是抓取一个网站上的股票代码，二是获取另外一个网站上的股票利好新闻。

见示例 10-8。

```
import requests
from bs4 import BeautifulSoup
#获取网页或网页文件的内容
def getHTMLText(url):
    try:
        r = requests.get(url)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text
    except:
        return ""
#获取股票代码
def getStockList(lst, stockURL):
    html = getHTMLText(stockURL)
    soup = BeautifulSoup(html, 'html.parser')
    a = soup.find_all('a')
    for i in a:
        try:
            if '(' in i.string:           #包含符号()是股票代码
                lst.append(i.string)      #增加到股票列表
        except:
            continue
#1.获取股票代码的演示
stock_list_url = 'http://quote.eastmoney.com/stocklist.html'
slist=[]                                #股票代码列表
getStockList(slist, stock_list_url)
print('显示获取网页的股票代码: ')
```

```
print(slist)                                #显示获取网页的股票代码
#2. 获取新闻利好的演示
url2='http://stock.stockstar.com/list/4957.shtml'
html = getHTMLText(url2)
soup = BeautifulSoup(html, 'html.parser')
stockInfo = soup.find('div', attrs={'class': 'listnews'})
a = stockInfo.find_all('a')
print('\n股票新闻利好: ')
for i in a:
    try:
        href = i.attrs['href']
        print(i.string)
    except:
        continue
```

程序运行结果如下, 因为数据很多, 我们只显示部分数据。

```
.....
, '顶固集创(300749)', '宁德时代(300750)', '迈为股份(300751)', '隆利科技
(300752)', '爱朋医疗(300753)', '华致酒行(300755)', '中山金马(300756)', '罗博特
科(300757)', '七彩化学(300758)', '康龙化成(300759)', '迈瑞医疗(300760)']
```

股票新闻利好:

华铭智能: 筹划发行股份收购聚利科技 100%股权

东信和平: 2018 年净利 3892 万元 同比增长 3.15%

汉缆股份: 投资设立中东全资子公司

浙江东方: 拟出售海康威视及华安证券股票

上海天洋: 2018 年净利预增 20%-35%

嘉泽新能: 2018 年净利预增 58%-70%

宝鹰股份: 拟斥资 2.1 亿至 4.2 亿元回购股份

```
.....
```

本章介绍了有关金融量化方面的综合知识。读者可以根据自己的经验及对投资的感悟, 来用 Python 辅助进行投资分析和回测研究。

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：(010) 88254396; (010) 88258888

传 真：(010) 88254397

E-mail : dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

电子工业出版社总编办公室

邮 编：100036



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY